Technical Document 2898
February 1996

Terrain Parabolic Equation Model (TPEM) Version 1.5 User's Manual

A. E. Barrios

Naval Command, Control and
Ocean Surveillance Center
RDT&E Division

San Diego, CA
92152-5001

Approved for public release; distribution is unlimited.

Technical Document 2898
February 1996

Terrain Parabolic Equation Model (TPEM) Version 1.5 User's Manual

A. E. Barrios

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER RDT&E DIVISION
San Diego, California 92152-5001

K. E. EVANS, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

The work detailed in this report was performed for the Office of Naval Research by the Naval Command, Control and Ocean Surveillance Center RDT&E Division, Tropospheric Branch, Code 543. Funding was provided under program element 0602435N.

Released by
R. A. Paulus, Head
Tropospheric Branch

Under authority of
J. H. Richter, Head
Ocean and Atmospheric
Sciences Division

SB

CONTENTS

Appendices

Figures

Tables

BEFORE YOU BEGIN

This document describes the Terrain Parabolic Equation Model (TPEM) Version 1.5, its operation, and the format required for the environmental and system input files. TPEM 1.5 calculates and plots propagation loss in dB on a height vs. range display. It allows for range-dependent refractivity environments and variable terrain. TPEM 1.5 is based on methods and source code originally developed by Fred Tappert, from the University of Miami, for propagation over a smooth surface. It is a pure parabolic equation (PE) model based on the split-step Fourier method and is described in Barrios (1994), with a more efficient method described in Barrios (1993).

This program was developed at the Naval Command, Control and Ocean Surveillance Center RDT&E Division, Code 543, 49170 Propagation Path, San Diego CA 92152-7385. For technical support, call (619) 553-1429, DSN 553-1429, or FAX (619) 553-1417, between the hours of 7:15 a.m. and 4:45 p.m., Pacific Time, Monday through Thursday, excluding holidays. For support via the Internet, email to barrios@nosc.mil. The program is written entirely in FORTRAN using the Microsoft 32-bit FORTRAN Powerstation compiler. To facilitate distribution via the Internet, TPEM 1.5 has been compressed into one self-extracting file, TPEM15.EXE, which can be directly downloaded from NRaD's Ocean and Atmospheric Sciences Division (Code 54) web page, http://sun-spot.nosc.mil. System requirements for TPEM 1.5 are IBM/PC or compatible with an EGA or VGA graphics adapter and at least 4 MB of extended memory in addition to DOS.

INSTALLATION

Use the following procedure to install the software.

1. Create a directory named TPEM.
2. Copy the font file MSFONTS.FON from the distribution disk to the TPEM directory. This is the recommended font file, but any .FON file supplied with MS WINDOWS will work.
3. Go to the AUTOEXEC.BAT file and include the line 'SET FONT = C:\TPEM\MSFONTS.FON' to set the environment variable.
4. Ensure that the file DOSXMSF.EXE (32-bit DOS extender that allows an executable program created under FORTRAN Powerstation to execute under DOS) is in the same directory as TPEM.EXE or in your path.
5. Type TPEM15 to expand the TPEM15.EXE file.

NOTE

> Do not try to expand the TPEM15.EXE file while your current drive is the floppy drive. Once expanded, TPEM.EXE may be copied to a floppy diskette and then executed. After expansion, the TPEM15.EXE file may be deleted to conserve disk space.

Table 1 lists and describes the files contained in TPEM15.EXE.

Table 1. TPEM15.EXE file descriptions.

File	Description
README.1ST	A text file listing all files, and their descriptions, contained in TPEM15.EXE.
USERSMAN.TXT	Text file of TPEM 1.5 User's Manual documentation (no figures).
DOSXMSF.EXE	Fortran Powerstation extender (required to be in path to execute TPEM 1.5).
MSFONTS.FON	Recommended Microsoft font file to be set in FONT environment variable in AUTOEXEC.BAT file.
TPEM.EXE	TPEM 1.5 DOS executable program. Requires extended memory.
SAMPLE.FOR	Fortran source code for sample driver program that calls routines in TPEMSUBS.FOR.
TPEMSUBS.FOR	Fortran source code containing all calculation routines used in TPEM 1.5.
SINFFT.FOR	Fortran source code for sine Fast Fourier Transform (FFT) routine. The include file, FFTSIZ.INC, is required.
TPEM.INC	Fortran source include file used in TPEM.EXE and required for SAMPLE.FOR and TPEMSUBS.FOR routines. Contains all variable information to be initialized by main driver program.
FFTSIZ.INC	Fortran source include file used in TPEM.EXE and required for SAMPLE.FOR and TPEMSUBS.FOR routines. Contains variable information for size limits on FFT transform arrays. This file, in conjunction with SINFFT.FOR, can be used as a stand-alone sine FFT routine.
SAMPLE.OUT	ASCII file containing output data from program SAMPLE.FOR.
TPEM.INP	Sample input file.
STAND.MET	Sample refractivity profile consisting of a homogeneous standard atmosphere.
300MSBD.MET	Sample refractivity profile consisting of a homogeneous 300-m surface-based duct.
RANGDEP.MET	Measured range-dependent refractivity profile (Barrios, 1994).
WEDGE.TER	Sample terrain profile consisting of a wedge 10 km wide, 200 m high, and centered at 50 km from starting range.
LONGBMU.TER	Terrain profile for path from Long Beach to Point Mugu, CA.
RADAR.SYS	Sample radar system file containing default parameters used in EREPS 3.09 - COVER.

OPERATION

Starting TPEM without any arguments on the command line will print a title page consisting of the program name, author, version, date, and usage summary. To run TPEM, there are several options. The format of the command line string follows DOS conventions. Depending on the option specified, however, the command line must follow certain formats. Options are case-insensitive.

In the command line options and formats described below, terms in *italics* are place holders and imply that you must specify a value or file name following DOS conventions. Table 2 lists the definitions of the command line parameters.

Table 2. Command line parameter definitions.

Parameter	Definition
infile	name of input file
envfile	name of refractivity file
terfile	name of terrain file
outfile	name of ascii output file
sysfile	name of radar system file
xr	range of output point
yh	height of output point

The format of the files in table 2 is described in the section **File Formats**. Below is a list of allowable command line formats, with options, and their descriptions.

1. **TPEM [/b] [/v]** *infile*

   /b **Batch mode.** Useful for batch runs. With this option, after a coverage diagram is displayed, the program automatically exits to the DOS prompt. Parameter *infile* must be specified.

   /v **View mode.** Useful for viewgraphs. Uses the labels, LOC, DATE, and TIME (in the refractivity data file) for printing on the right-hand side of the screen. Parameter *infile* must be specified.

   Example command line:

   TPEM TPEM.INP

   This is the most basic run of TPEM. Invoking this command will produce a coverage diagram with the parameters specified in the file TPEM.INP.

   Example command line:

   TPEM /b TPEM.INP

   This produces a coverage diagram with parameters specified in TPEM.INP and exits back to the DOS prompt. In a typical use, this would be just one command line of several in a batch file in which TPEM was executed with several different input files.

2. **TPEM /p [/b] [/v]** *infile sysfile*

   /p Propagation loss is displayed as probability of detection contours. Parameters *infile* and *sysfile* must be specified.

Example command line:

TPEM /p TPEM.INP RADAR.SYS

This produces a coverage diagram in which loss thresholds are determined from radar system information given in RADAR.SYS and 10% to 90% probability of detection.

3. **TPEM /f** *infile xr yh envfile outfile*

   /f Useful for obtaining propagation loss at a specified range and height (*xr* and *yh*, respectively) over a smooth (no terrain) surface. Parameters on the command line that must be specified are *infile*, *xr*, *yh*, *envfile*, and *outfile*. The environment file, *envfile*, specified on the command line will override the file specified in *infile*. This option automatically runs in batch mode, so multiple runs can be made where the propagation loss determined at different values of *xr* and *yh* are stored in ASCII in the file parameters *outfile*. Parameters *xr* and *yh* are taken to be in the same units specified in the file *infile*.

   Example command line:

   TPEM /f TPEM.INP 100. 30.5 300MSBD.MET DATAOUT

   This produces a coverage diagram with the parameters specified in TPEM.INP, but with refractivity specified by 300MSBD.MET. In addition, if the units used are metric, the propagation loss at a range of 100 km and a height of 30.5 m above mean sea level is stored in the ASCII file DATAOUT.

4. **TPEM /t** *infile xr yh envfile terfile outfile*

   /t Same as the **/f** option, but allows for terrain runs. Parameters on the command line that must be specified are *infile*, *xr*, *yh*, *envfile*, *terfile*, and *outfile*. The environment and terrain files specified on the command line will override the files specified in *infile*. This option automatically runs in batch mode, so multiple runs can be made where the propagation loss determined at different values of *xr* and *yh* are stored in the ASCII file *outfile*. Parameters *xr* and *yh* are taken to be in the same units specified in the file *infile*. NOTE: The height specified by *yh* is assumed to be the height above the local terrain at range *xr*.

   Example command line:

   TPEM /t TPEM.INP 100. 30.5 300MSBD.MET LONGBMU.TER DATAOUT

   This produces a coverage diagram with the parameters specified in TPEM.INP, but with the refractivity and terrain files specified by 300MSBD.MET and LONGBMU.TER on the command line. In addition, if the units used are metric, the propagation loss at a range of 100 km and a height of 30.5 m above the ground (at 100 km) is stored in the ASCII file DATAOUT.

If TPEM is executed under any option other than the **/b**, **/f**, or **/t** options, changes to the input parameters in *infile* can be made, and TPEM re-executed, without exiting the program. Once a coverage diagram has been displayed, simply type **ALT+P** and you are automatically placed in a "prompts" page with the cursor at the top (figure. 1).

From this page, all the parameters specified in *infile* can be changed by simply editing each prompt input. The arrow keys are used to move up or down each prompt. Once all changes are made, press the **F10** key and a new coverage diagram will be displayed with the new input parameters. In addition, while in the "prompts" page, the title can be changed by pressing **ALT+T** and simply typing an alternate title, followed by **ENTER**.

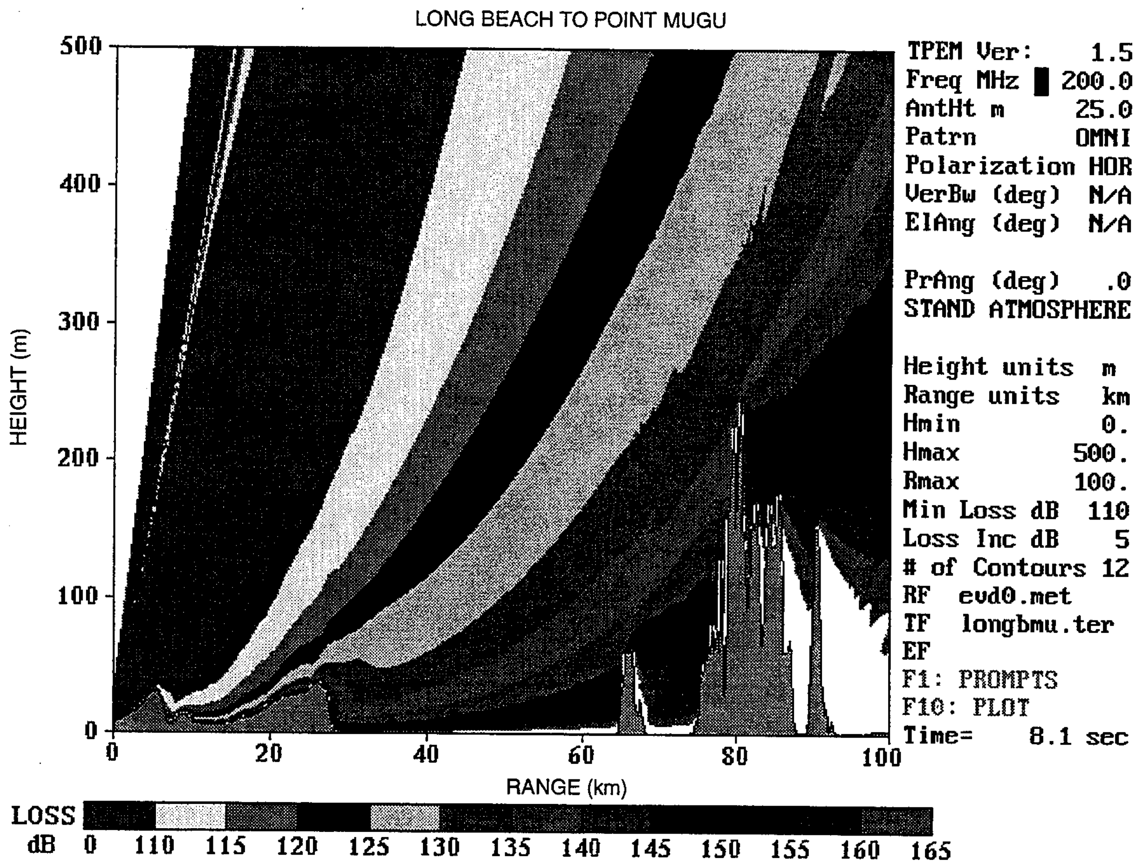

Figure 1. Coverage diagram for Long Beach to Point Mugu terrain path, standard atmosphere.

While in the "prompts" page, the lines RF, TF, and EF refer to the refractivity file name, terrain file name, and EREPS-compatible file name, respectively. At these prompts you can enter a new refractivity file, terrain file, or EREPS binary file for storing output. Inputs for the antenna pattern, polarization, height units, and range units, can be changed by pressing the **SPACE** bar. The parameters for these prompts will then be toggled through their available options. To eliminate unnecessary information from the right-hand side of the screen (for cleaner hardcopies or viewgraphs), press **F1**. This will toggle the graphics input information on the lower half of the right-hand-side of the screen.

FILE FORMATS

In the following files, comments are specified by a pound sign (#) in the first column of a line. There is no limit to the number of comment lines in a file, as they are ignored by TPEM. All files are assumed as ASCII and file names follow DOS conventions.

INPUT FILE (*INFILE*)

An example input file is shown in figure 2. All input parameters must be followed by a colon. All text to the right of the colon is assumed to be a comment and is ignored by TPEM. Antenna patterns are specified by typing only the first letter of the desired pattern. Due to numerical constraints, the maximum plot height is limited to no less than 100 m and the maximum plot range is limited to no

less than 5 km. The maximum propagation angle is an optional input that allows you to override the internal calculation by TPEM. If 0, TPEM will automatically calculate a maximum angle such that coverage is obtained at all heights and AT LEAST 90% of the maximum plot range. If you wish to see greater coverage (at higher angles), then you can specify any value (up to a maximum of 15 degrees) and TPEM will use this angle. Specifying a file name in the last line of the input file will produce an EREPS-compatible binary file that contains all the propagation loss information of the coverage diagram. This file can be read by EREPS (dated on or after 31 January 1996) to display loss vs. range at any receiver height, or height vs. loss at any receiver range. For the format of this binary file, refer to EREPS 3.0 documentation (Patterson, 1994).

```
#TPEM Version 1.5 system input file
M            : Height units (M = meters, FT = feet)
KM           : Range units (M,KM=kilometer,NM=nautical mile,SM=statute mile)
200          : Frequency in MHz (100 to 20000)
25           : Transmitter height in above units
O            : Ant pattern(Omni, Gaussian, Sin(x)/x, Csc-Sq, Ht-finder)
H            : Polarization (H-horizontal, V-vertical)
3            : Beamwidth in degrees (full 3 dB to 3 dB width)
0            : Elevation angle in degrees
0.           : Maximum propagation angle in degrees.
0.           : Minimum height with respect to m.s.l. in above units
500.         : Maximum height with respect to m.s.l. in above units
100.         : Maximum range in above units
110          : Minimum propagation loss in dB for color scale
5            : Propagation loss increment in dB for color scale
12           : Number of colors (or contours) for coverage diagram
300msbd.met : DOS file name for environment (refractivity profiles)
wedge.ter    :DOS file name for terrain profile (Blank assumes smooth surface)
             :DOS file name for EREPS output. Blank gives no EREPS output.
```

Figure 2. Example input file.

ENVIRONMENT FILE (*envfile*)

An example refractivity environment file is shown in figure 3. All refractivity profiles must be specified in terms of height/M-unit pairs, where the first column contains the height at each refractivity level and the second column contains the M-unit value at that level. The first refractivity profile must always be at range 0. For range-dependent environments, subsequent profiles can be added following the same rules. Each subsequent profile must be preceded by the RANGE keyword and range value for that particular profile, and all profiles must have the same number of refractivity levels. All height levels in refractivity profiles are assumed to be relative to mean sea level. There are several key words in the environment files that are recognized by TPEM. All keywords must be preceded by the @ symbol in the first column of each line, and must be followed by a space and the information associated with that keyword. Table 3 lists environment file keywords and their definitions.

```
#Environment file for TPEM 1.5 - Standard Atmosphere

@label Stand Atm
@range 0. km
@height m

0.    0.
1000. 118.
```

Figure 3. Example environment file.

Table 3. Environment file keywords.

Keyword	Definition
LABEL (optional)	Indicates a 16-character descriptor of the environment is to follow. Used as the environment label to be printed on the right-hand side of the screen display.
LOC (optional)	Indicates a 12-character descriptor of date of refractivity profile to follow. Used only with the "viewgraph" or /v option.
DATE (optional)	Indicates a 12-character descriptor of date of refractivity profile to follow. Used only with the "viewgraph" or /v option.
TIME (optional)	Indicates a 12-character descriptor of date of refractivity profile to follow. Used only with the "viewgraph" or /v option.
RANGE (required)	Followed by a number and a character units descriptor indicates the range and units of the refractivity profile. Acceptable unit descriptors are KM (kilometers), NM (nautical miles), and SM (statute miles).
HEIGHT (required)	Followed by a units descriptor indicates the units of height levels of the profile. Acceptable unit descriptors are M (meters) and FT (feet).

TERRAIN FILE (*terfile*)

An example terrain file is shown in figure 4. All terrain profiles must be specified in terms of range/height pairs, where the first column contains the range and the second column contains the terrain elevation, or height, at that range. All range values must be increasing, and the first terrain elevation value must be at range 0. TPEM allows a maximum of 1000 range/height pairs. A warning will occur if this is exceeded. The format for keywords is the same as in the refractivity file, with the exception of the RANGE and HEIGHT keywords. These must be given on the same line (also preceded by an @ symbol in the first column) with each keyword followed by a colon. Terrain file keywords are described in table 4.

```
#TPEM 1.5 terrain file - Wedge centered at 50 km. - 200 m high and 10
#km wide. The first 45 km of the terrain path is sea water. From 45 km
#to 55 km the dielectric properties of the ground are permittivity = 50,
#conductivity = 1.e-4. The remainder of the path consists of sea water.

@label Wedge centered at 50 km
@ground sea water 0.
@ground userdef (50,1.e-4) 45.
@ground sea water 55.

@range: km    height:  m
0.                     0.
45.                    0.
50.                    200.
55.                    0.
100.                   0.
```

Figure 4. Example terrain file.

Table 4. Terrain file keywords.

Keyword	Definition
LABEL (optional)	Followed by a space and an 80-character string describing the terrain profile, i.e., latitude, longitude, city, country, etc. This will be centered and printed at the top of the coverage diagram.
RANGE (required)	Followed by a character units descriptor, indicates the units of range values to follow. Acceptable unit descriptors are: M (meters), KM (kilometers), NM (nautical miles), SM (statute miles).
HEIGHT (required and must be specified on the same line as RANGE keyword)	Followed by a character units descriptor, indicates the units of following height values. Acceptable unit descriptors are M (meters), FT (feet).
GROUND (required for vertical polarization)	Used to specify a particular ground type in order to model finite conductivity and vertical polarization, and must be followed by a ground-type descriptor. The following ground-type descriptors are allowed: SEA WATER, FRESH WATER, WET GROUND, MEDIUM DRY, VERY DRY, USERDEF. The keyword USERDEF signifies that you will define the ground type, in which case numerical values for the permittivity and conductivity must be separated by a comma and enclosed in parentheses. Following the descriptor, the range at which the ground type is to be applied must be specified. The GROUND keyword can be specified several times with different descriptors and ranges, indicating variable dielectric ground properties with range.

SYSTEM FILE (*sysfile*)

An example radar system file is shown in figure 5 for use with the **/p** option in producing coverage diagrams in terms of probability of detection contours. The numeric parameters illustrated are the

defaults used in EREPS 3.09 in the program COVER. The quantities in brackets at the end of each line refer to the numerical bounds for that particular parameter. If a value is entered outside of these bounds, then TPEM will produce an error and abort. For an explanation of what is meant by "simple" or "integrated" calculations, refer to the EREPS 3.0 documentation. As with the input file, all input parameters must be followed by a colon on the same line.

```
#TPEM 1.5 Radar system file with default parameters used in EREPS 3.09 -
#COVER.
S      : Radar calculation type; S-Simple, I-Integrated
I      : Coherent or Incoherent integration (only for I radar calc.)
1      : Number of pulses ( only for I radar calc )
21     : Antenna Gain (dB) [0 to 100]
11     : Horizontal Beamwidth (deg) [>0 to 90]
6      : Horizontal scan rate (rpm) [1 to 1000]
200    : Peak Power (kW) [.1 to 1.e4]
60     : Pulse width (microsec) [.1 to 1.e4]
300    : Pulse repetition frequency (pps) [1 to 1.e4]
6      : System loss (dB) [0 to 100]
5      : Noise figure (dB) [0 to 100]
10.    : Radar cross section (sqm) [>0 to 1.e5]
1.e-8 : Probability of false alarm [1.e-4 to 1.e-12]
1      : Swerling case 0-steady, 1-fluctuating
```

Figure 5. Example radar system file.

SUMMARY OF CAPABILITIES

1. Accommodates up to 30 M-unit vs. height profiles at arbitrary ranges, with up to 300 refractivity/height levels for each profile. The first profile must be at range zero.
2. Accomodates any general terrain profile up to a maximum of 1000 height/range pairs. First point must be at range zero.
3. Frequency is from 100 MHz to 20 GHz.
4. Allows horizontal and vertical polarization with user-defined dielectric ground properties.
5. Antenna height is variable up to the available memory limit specified by maximum transform size.
6. Antenna patterns: Omni, Gaussian, Sin(x)/x, Cosecant-squared, or generic height-finder.
7. Vertical beamwidth: 0.5 to 45 degrees.
8. Antenna elevation angle: -10 to 10 degrees.
9. No maximum limits on plot range, but is restricted to no less than 5 km.
10. Maximum height is limited to available transform size and is restricted to no less than 100 m.

COMMENTS

For range-dependent refractivity environments, if the maximum plot range given is greater than the range of the last profile, a warning is given and you are prompted to continue the run. If you answer **Y**(es) (type '**Y**', but do not press **ENTER),** TPEM will produce a coverage diagram out to the maximum range specified, with the refractivity environment taken to be homogeneous between the range of the last profile and the maximum range. If you answer **N**(o), the run is aborted.

If the number of points in the terrain profile exceeds the maximum allowed, a warning is given and you are prompted to continue the run. If you answer **Y**(es), TPEM will ignore the remaining points in the terrain profile and will use only those range/height pairs up to the 1000-point maximum allowed. If you answer **N**(o), the run is aborted.

If the last range point in the terrain profile is less than the maximum plot range, a warning is given and you are prompted to continue the run. If you answer **Y**(es), TPEM will assign the last point in the profile to be equal to the maximum plot range and the elevation height to be equal to the last height given in *terfile*.

All antenna pattern and sidelobe effects are handled as in EREPS 3.09. For details, refer to the EREPS 3.0 documentation.

TPEM is a pure split-step PE model, and therefore, coverage is angle- and height-limited. TPEM is designed to automatically calculate an angle, given the input parameters, such that coverage is obtained at all heights from 90% of the maximum range onward. At lesser ranges, coverage will then be reduced in height. If you wish to see greater coverage, i.e., higher angles, the maximum propagation angle can be specified with a non-zero (positive) value in *infile*.

TPEM will only model finite conductivity when vertical polarization is specified. Therefore, even if the GROUND keyword with proper descriptors is included in the terrain profile, perfect conductivity will be assumed for horizontal polarization. For all ground types other than USERDEF, the permittivity and conductivity are calculated as a function of frequency from curve fits to the permittivity and conductivity graphs shown in recommendataions and reports of the International Radio Consulting Committe (1986).

All coverage diagrams are displayed relative to the minimum height you specify in *infile*. Also, all loss values stored in EREPS-compatible binary files will be relative to this height. Coverage will only be displayed up to the smaller of the maximum height you enter, or the maximum height determined by the given problem and maximum transform size.

The antenna height entry in the input file always refers to the antenna height above the local ground at range 0.

To read EREPS-compatible binary files created by TPEM, obtain EREPS30.EXE (dated after 31 January 1996) from the Internet at the address, http://www.sunspot.nosc.mil/543/software.html. This is a self-extracting file that creates the individual EREPS programs (PROPR, PROPH, COVER, etc.) that can then be used to read TPEM binary files.

When reading TPEM binary files through PROPR, loss is plotted vs. range at a specified height. This height is relative to the minimum height you originally specified (in *infile*) to create the file.

SOURCE CODE IMPLEMENTATION

This section describes the source code implementation of the main calculation routines used in TPEM 1.5. All the source routines are contained in the file TPEMSUBS.FOR and are written using the 32-bit Microsoft FORTRAN Powerstation compiler.

In order to develop your own application using the TPEM model, you must create a main driver program that calls the routines in TPEMSUBS.FOR. A sample driver program (SAMPLE.FOR) is provided and will be discussed in the next section of this document. You need to call only two subroutines from the main driver program: PEINIT and PESTEP. PEINIT initializes and processes all the information that is passed to it by the main driver program. PESTEP is then called by the driver program at each range step to propagate the field forward and return the propagation loss at specified height output points.

SAMPLE.FOR

Included within TPEM15.EXE is a sample driver program called SAMPLE.FOR. In order to make a stand-alone executable program that calculates propagation loss at specified height and ranges over variable terrain, the following files need to be compiled and linked:

TPEM.INC
FFTSIZ.INC
SAMPLE.FOR
TPEMSUBS.FOR
SINFFT.FOR

SAMPLE.FOR determines loss values at heights of 200 m, 400 m, 600 m, 800 m, and 1000 m at each of these ranges: 20 km, 40 km, 60 km, 80 km, and 100 km. The environment consists of a homogeneous 300-m surface-based duct with a wedge-terrain profile 10 km wide and 200 m high centered at 50 km. The frequency is 1000 MHz. The antenna is omnidirectional and the antenna height is at 25 m. The ASCII output is written to a file called SAMPLE.OUT, also included within TPEM15.EXE and listed in appendix A.

Initialization of all information you must supply to the model will be discussed below. In the following subroutine calls, variables in parameter lists in lowercase letters are variables that are passed to the called routine, while those in uppercase letters are variables that are returned.

PEINIT

In order for this routine to initialize and process information for subsequent calls to PESTEP, information must be passed to it from the main driver program. The variables that need to be initialized from the main program are in the include file, TPEM.INC, listed in appendix B. Parameter statements are also included here that set the maximum amount of points in a refractivity profile, a terrain profile, etc. These constants are used in TPEM 1.5, however, you can change these values to suit your particular application.

Since TPEM is a pure split-step PE model, coverage will not be given at all heights and ranges specified. PEINIT automatically determines a suitable angle such that coverage is given to AT LEAST all heights and 90% of the maximum range or greater. You can always overide this by specifying an angle for desired coverage in the variable PROPANG (in degrees) in the INPUTVAR structure.

All variables are described in TPEM.INC. Once these are initialized, the call to PEINIT follows:

call peinit(ef, vnp, rf, sv, tr, HMINTER, ROUT, IERROR)

where EF, VNP, RF, SV, and TR are structures (from TPEM.INC) declared with the following statements in the main driver program:

```
record / errorflag / ef
record / inputvar / vnp
record / refractivity / rf
record / systemvar / sv
record / terrain / tr
```

Upon exit, PEINIT returns HMINTER, ROUT, and IERROR. HMINTER is the reference height that is determined from the terrain profile passed in the structure TR. It corresponds to the minimum terrain elevation in the terrain profile. The terrain profile is adjusted by this height in order to maximize the calculation domain. This is the reference height for all internal field calculations. ROUT is the current range step in meters and is initialized to 0.0 upon exiting PEINIT in preparation for subsequent calls to PESTEP. IERROR is an integer error flag that returns a negative value if an error occurs in PEINIT. Error codes are returned to ensure that the information input to the model is valid and within the model's designed limits. If these errors are ignored, erroneous solutions may result. Of course, not all parameters are fully checked, but the errors listed below are the most important, which, if left ignored, may cause your program to abort. Table 5 lists returned error values and their definitions.

Table 5. Error value definitions.

IERROR	Definition
–6	Last range in terrain profile is less than RMAX. (Will only return this error if error flag EF.LERR6 is set to .TRUE).
–8	HMAX is less than maximum height of terrain profile.
–12	Range of last refractivity profile entered (for range-dependent case) is less than RMAX. (This is returned from subroutine REFINIT). Will only return this error if error flag EF.LERR12 is set to .TRUE).
–14	Last gradient in any refractivity profile entered is negative. (This is returned from REFINIT).
–17	Range points in terrain profile are not increasing.
–18	First range point is not 0.

PESTEP

Once the input data has been initialized and PEINIT returns no error, calls to PESTEP are made to determine propagation loss at specified range steps. This call is

call pestep(hminter, vnp, rf, tr, sv, ROUT, MLOSS, JSTART, JEND)

MLOSS is a 2-byte integer array (must be declared in main driver program) that contains the propagation loss in centibels, i.e., MLOSS() = NINT(propagation loss in dB * 10.). JSTART is the index at which valid loss points begin in MLOSS(). JEND is the index at which valid loss points end in

MLOSS(). The values in MLOSS() will always be referenced to the minimum height you specify via the HMIN variable in structure INPUTVAR.

Terrain information is also contained within MLOSS(). All loss values returned in MLOSS() with a value of 0 represent terrain, while all loss values of -1 represent invalid loss data. For example, if a terrain profile is specified, and the number of output height points specified is 5 (NZOUT=5, in structure INPUTVAR), then for a given output range step less than 90% of RMAX, MLOSS() may be returned as:

MLOSS(1) = 0 ⇒ terrain height at this range is at least of height DZ.
MLOSS(2) = loss1 ⇒ propagation loss in centibels at height 2.*DZ
MLOSS(3) = loss2 ⇒ propagation loss in centibels at height 3.*DZ
MLOSS(4) = loss3 ⇒ propagation loss in centibles at height 4.*DZ
MLOSS(5) = -1 ⇒ propagation angle is greater than maximum angle in PE solution at height 5.*DZ, therefore, there is an invalid loss solution at this height.

In this case, JSTART will have a value of 2 (since this is the start of valid loss data), and JEND will have a value of 4 (since this is the end of valid loss data). Here, DZ is the output height increment given by DZ=(HMAX-HMIN)/NZOUT, with all loss values referenced to height HMIN.

REFERENCES

Barrios, A. E. 1994. "A Terrain Parabolic Equation Model for Propagation in the Troposphere," *IEEE Trans. on Ant. and Prop.*, vol. 42, no. 1 (Jan), pp. 90-98.

Barrios, A. E. 1993. "Terrain and Refractivity Effects on Non-Optical Paths," *AGARD Conference. Proceedings 543, Multiple Mechanism Propagation Paths (MMPPs): Their Characterization and Influence on System Design* (pp. 10-1 to 10-9). October.

International Radio Consulting Committee (CCIR). 1986. "Propagation in Non-Ionized Media," *Recommendations and Reports of the CCIR*, vol. V.

Patterson, W. L., C. P. Hattan, G. E. Lindem, R. A. Paulus, H. V. Hitney, K. D. Anderson, and A. E. Barrios. 1994. "Engineer's Refractive Effects Prediction System (EREPS) Version 3.0." NRaD TD 2648(May). Naval Command, Control and Ocean Surveillance Center RDT&E Division, San Diego, CA.

APPENDIX A

SAMPLE.FOR LISTING

```
c This is a sample driver program for TPEM routines PEINIT and PESTEP.
c All numeric parameters passed to PEINIT and PESTEP must be in metric
c units.

      program sample

      include 'tpem.inc'

      record / errorflag / ef
      record / inputvar / vnp
      record / refractivity / rf
      record / systemvar / sv
      record / terrain / tr

      integer*2 mloss(mxzout) !MLOSS must be declared an INTEGER*2 array
                              !of size at least MXZOUT.

c This is a 300 m surface-based duct.

      data (rf.refmsl(i,1),i=1,4) / 339., 368.5, 319., 401.6 /
      data (rf.hmsl(i,1),i=1,4) / 0., 250., 300., 1000. /

c This is a wedge terrain profile with the center of the wedge at a
c range of 50 km and a height of 200 m.  The base of the wedge
c spans a width of 10 km.

      data (tr.terx(i),i=1,5) / 0., 45000., 50000., 55000., 100000. /
      data (tr.tery(i),i=1,5) / 0., 0., 200., 0., 0. /

c Set logical flags to trap for errors.  LERR6=.TRUE.-PEINIT returns
c error if last point in terrain profile is less than maximum plot
c range.LERR12=.TRUE.-PEINIT return error if last refractivity profile
c entered(for range-dependent environment) is less than maximum plot
c range.

      ef.lerr6 = .true.
      ef.lerr12 = .true.

      vnp.hmin = 0.             !Minimum height is 0. m
      vnp.hmax = 1000.          !Coverage up to 1000. m.
      vnp.rmax = 100000.        !Range up to 100 km.
      vnp.nzout = 5.            !Output 5 height points.
      vnp.nrout = 5.            !Output 5 range points.
      vnp.propang = 0.          !Automatic internal angle calculation.

      rf.lvlep = 4              !Specify 4 levels in refractivity profile.
      rf.nprof = 1              !This is a range-independent case.
      rf.rngprof(1) = 0.        !Range of profile is at range 0.

      sv.freq = 1000.           !Frequency is 1000 MHz.
      sv.antht = 25.            !25 m antenna height.
      sv.ipat = 0               !Omni antenna.
      sv.polar = 'H'            !Horizontal polarization.
      sv.bwidth = 1.            !This value is ignored for Omni antenna.
      sv.elev = 1.              !This value is ignored for Omni antenna.

      tr.itp = 5                !5 range/height pairs in terrain profile
```

```
c Variables in CAPS are returned.

      call peinit( ef, vnp, rf, sv, tr, HMINTER, ROUT, IERROR )

      if( ierror .ne. 0 ) then
         write(*,*)'*********** ERROR IN PEINIT **************'
         write(*,*)'******** IERROR = ', ierror,' **********'
         return
      end if

      nr = vnp.nrout
      dz = (vnp.hmax-vnp.hmin) / float( vnp.nzout ) !Determine height
                                                    !increment of
                                                    !output points.
      open( 15, file='sample.out' )

c The reference height in this case is 0. since the minimum height in
c the terrain profile is 0.

      write(15,*)'Reference height in m = ', hminter

      do i = 1, nr

         call pestep( hminter, vnp, rf, tr, sv, ROUT, MLOSS, JSTART,
     +                JEND )

         write(15,*)
         write(15,*)'range in km = ', rout*1.e-3
         write(15,*)
         write(*,*)'range in km = ', rout*1.e-3  !Output to screen

c Recall that MLOSS is the propagation loss in centibels, i.e.,
c MLOSS() = NINT( propagation loss in dB * 10. ).  JSTART = start of
c valid loss points, JEND = end of valid loss points.

         do j = jstart, jend
            write(15,*)'Height (m)= ',j*dz,'  loss in dB = ',mloss(j)*.1
         end do

      end do

      close(15)
      end
```

SAMPLE.OUT

```
Reference height in m =    0.000000E+00

 range in km =       20.000000

 Height (m)=       200.000000  loss in dB =        112.500000
 Height (m)=       400.000000  loss in dB =        114.100000
 Height (m)=       600.000000  loss in dB =        130.700000
 Height (m)=       800.000000  loss in dB =        113.300000
 Height (m)=      1000.000000  loss in dB =        115.200000

 range in km =       40.000000

 Height (m)=       200.000000  loss in dB =        119.400000
 Height (m)=       400.000000  loss in dB =        119.200000
 Height (m)=       600.000000  loss in dB =        118.700000
 Height (m)=       800.000000  loss in dB =        121.300000
 Height (m)=      1000.000000  loss in dB =        133.700000

 range in km =       60.000000

 Height (m)=       200.000000  loss in dB =        126.000000
 Height (m)=       400.000000  loss in dB =        127.500000
 Height (m)=       600.000000  loss in dB =        123.500000
 Height (m)=       800.000000  loss in dB =        130.000000
 Height (m)=      1000.000000  loss in dB =        123.400000

 range in km =       80.000000

 Height (m)=       200.000000  loss in dB =        132.900000
 Height (m)=       400.000000  loss in dB =        137.600000
 Height (m)=       600.000000  loss in dB =        128.900000
 Height (m)=       800.000000  loss in dB =        126.400000
 Height (m)=      1000.000000  loss in dB =        136.600000

 range in km =      100.000000

 Height (m)=       200.000000  loss in dB =        145.300000
 Height (m)=       400.000000  loss in dB =        148.800000
 Height (m)=       600.000000  loss in dB =        138.100000
 Height (m)=       800.000000  loss in dB =        131.700000
 Height (m)=      1000.000000  loss in dB =        128.500000
```

APPENDIX B

PROGRAM LISTING

FFTSIZ.INC INCLUDE FILE

```
c MXNFFT:  Maximum power of 2 for transform size
c MAXPTS:  Maximum size of arrays for the real and imaginary fields

      integer*4 maxpts, mxnfft
      parameter ( mxnfft = 14)
      parameter ( maxpts = 2**mxnfft )
```

TPEM.INC INCLUDE FILE

```
include 'fftsiz.inc'

      integer*4 maxn4, mxzout, mxrout, mxlvls, mxnprof, mxter

      parameter ( maxn4 = maxpts/4 )  !used for filter array - filters
                                      !upper 1/4 of field.
      parameter ( pi = 3.1415926 )  !Self-explanatory
      parameter ( mxzout = 385 )   !Maximum number of output height points
      parameter ( mxrout = 440 )   !Maximum number of output range points
      parameter ( mxlvls = 300 )   !Maximum number of height/M-unit levels
      parameter ( mxnprof = 30 )   !Maximum number of profiles allowed for
                                   !range-dependent environment.
      parameter ( mxter = 1002 )   !Maximum number of height/range points
                                   !allowed for terrain profile

c ERRORFLAG:
c  LERR6 =  Logical flag that allows for greater flexibility in allowing error
c           -6 to be bypassed.  If set to .TRUE. then trapping for this error
c           occurs, otherwise it can be totally ignored by main driver
c           program.
c           (Within the TPEM program it is handled as a warning).  If this
c           error is bypassed (LERR6 = .FALSE.) terrain profile is extended to
c           RMAX with same elevation height of last valid terrain profile
c           point.
c  LERR12 = Same as LERR6 - allows for trapping of this error.  If LERR12 =
c           .FALSE., then (for range-dependent case) if range of last
c           refractivity profile entered is less than RMAX, the environment
c           is treated as homogeneous from the last profile entered to RMAX.

      structure / errorflag /
         logical lerr6
         logical lerr12
      end structure

c INPUTVAR:
c   HMAX = maximum output height with respect to m.s.l. in meters
c   HMIN = minimum output height with respect to m.s.l. in meters
c   RMAX = maximum output range in meters
c   NZOUT = integer number of output height points desired
c   NROUT = integer number of output range points desired
c   PROPANG = Maximum problem (propagation) angle in degrees desired for
c             solution. If set to 0., then TPEM will determine it's own.

      structure / inputvar /
         real hmax
         real hmin
         real rmax
         integer*4 nzout
         integer*4 nrout
         real propang
      end structure

c REFRACTIVITY:
c   LVLEP = number of levels in refractivity profile (for range dependent case
```

```
c          all profiles must have same number of levels)
c   REFMSL() = 2-dimensional array containing refractivity with respect
c              to mean sea level of each profile.  Array format must be
c   REFMSL(I,J) = M-unit value at Ith level of Jth profile.  J = 1 for range-
c                 independent cases.
c   HMSL() = 2-dimensional array containing heights in meters with respect to
c            mean sea level of each profile.  Array format must be HMSL(I,J) =
c            height of Ith level of Jth profile.  J = 1 for range-independent
c            cases.
c   RNGPROF() = ranges of each profile in meters, i.e., RNGPROF(I) = range of
c               Ith profile.  RNGPROF(1) should always be equal to 0.
c   NPROF = number of profiles.  Equals 1 for range-independent cases.

      structure / refractivity /
         integer*4 lvlep
         real refmsl(mxlvls, mxnprof)
         real hmsl(mxlvls, mxnprof)
         real rngprof(mxnprof)
         integer*4 nprof
      end structure

c SYSTEMVAR:
c   FREQ = frequency in MHz
c   ANTHT = transmitting antenna height above local ground in meters.
c   BWIDTH = half-power (3 dB) antenna pattern beamwidth in degrees (.5 to
c            45.)
c   ELEV = antenna pattern elevation angle in degrees. (-10 to 10)
c   POLAR = 1-character string indicating polarization.  H-horizontal,
c           V-vertical
c   IPAT = integer value indicating type of antenna pattern desired
c          IPAT = 0 -> omni
c          IPAT = 1 -> gaussian
c          IPAT = 2 -> sinc x
c          IPAT = 3 -> csc**2 x
c          IPAT = 4 -> generic height-finder

      structure / systemvar /
         real freq
         real antht
         real bwidth
         real elev
         character*1 polar
         integer*4 ipat
      end structure

c TERRAIN:
c  TERX() = range points of terrain profile in meters
c  TERY() = height points of terrain profile in meters
c  ITP = number of height/range pairs in profile
c  IGR = number of different ground types specified
c  IGRND() = type of ground composition for given terrain profile - can vary
c            with range.  Different ground types are: 0 = sea water,
c            1 = fresh water, 2 = wet ground, 3 = medium dry ground,
c            4 = very dry ground, 5 = user defined (in which case, values of
c            relative permittivity and conductivity must be given).
c  RGRND() = ranges at which the ground types apply
c  DIELEC(,) = 2-dimensional array containing the relative permittivity and
c              conductivity; DIELEC(1,i) and DIELEC(2,i), respectively.
c              Only needs to be specified if using IGRND(i) = 5, otherwise,
c              TPEM will calculate based on frequency and ground types 0-4.

      structure / terrain /
         real terx(mxter)
         real tery(mxter)
         integer*4 itp
         integer*4 igr
         integer*4 igrnd(50)
```

```
   real rgrnd(50)
   real dielec(2,50)
end structure
```

TPEMSUBS.FOR

```
c **************** THIS FILE CONTAINS TPEM MODEL SUBROUTINES ***************

c Author: Amalia E. Barrios
c         NCCOSC RDT&E DIV 543
c         49170 Propagation Path
c         San Diego, CA  92152-7385
c         e-mail: barrios@nosc.mil
c         phone: (619) 553-1429
c         fax: (619) 553-1417

c Summary: These routines model tropospheric radiowave propagation over
c          variable terrain and calculates propagation loss vs. height and
c          range. Propagation loss is displayed in dB contours on a height vs.
c          range plot. TPEM is based on the split-step Fourier PE method and
c          was originally developed from an early PE model called PEPC,
c          written by Fred Tappert.  Propagation loss over variable terrain is
c          modeled by shifting the field an appropriate number of bin widths
c          corresponding to the height of the ground.  The field is determined
c          using the smooth earth PE method.

c***************************************************************************

c Variables in small letters in parameter lists are variables that are input
c or passed to called subroutines.  Variables in CAPS in parameter lists are
c returned from the called subroutines.

c *************************** SUBROUTINE PEINIT *****************************

c Purpose:  Initializes all variables used in TPEM subroutines for PE calcula-
c           tions.

c Parameter list:

c HMINTER = Height of the minimum elevation of terrain profile.  This will be
c           used to adjust entire terrain profile so subsequent loss values
c           returned will be referenced to this height.
c ROUT = Output range point (meters) - initialized in this routine
c IERROR = Integer value that is returned if any errors exist in input data:
c             -6 : Last range in terrain profile is less than VNP.RMAX. (Will
c                  only return this error if error flag EF.LERR6 is set to
c                  .TRUE.).
c             -8 : VNP.HMAX is less than maximum height of terrain profile.
c            -12 : Range of last refractivity profile entered (for range depen-
c                  dent case) is less than RMAX. (This is returned from subrou-
c                  tine REFINIT).  Will only return this error if error flag
c                  EF.LERR12 is set to .TRUE.).
c            -14 : Last gradient in any refractivity profile entered is
c                  negative.(This is returned from REFINIT).
c            -17 : Range points of terrain profile is not increasing.
c            -18 : First range point is not 0.
c            -42 : Minimum height input by user (VNP.HMIN) is greater then
c                  maximum height (VNP.HMAX).

c Called from MAIN DRIVER PROGRAM

c Routines called: REFINIT, TRACEA, GETFFTSZ, XYINIT, SINFFT, TRACEH,
C                  PHASE1, PROFREF, INTPROF, PHASE2

      subroutine peinit( ef, vnp, rf, sv, tr, HMINTER, ROUT, IERROR )

      include 'tpem.inc'

c Common Blocks

c ARRAYS:
```

```
c   ENVPR() = Complex array containing refractivity exponential term.
c             i.e. ENVPR() = exp[i * dr * k * 1e-6 * M(z) ]
c   FILT() = Cosine-tapered (Tukey) filter array.
c   FRSP() = Complex array containing free-space propagator exponential term.
c            i.e., FRSP() = exp[-i * dr * (k - sqrt(k**2 - p**2)) ]
c   U() = Complex array containing field solution.
c   ULST() = Complex array containg field solution at previous range step.

c HTVAR:
c   YLAST = height of ground at the last range step
c   YCUR = height of ground at the current range step
c   YCURM = height of ground midway between last and current range step.
c           For use when shifting profiles to be relative to the local ground
c           height.

c IMPEDANCE:
c   ALPHAV = vertical polarization impedance term = i*fko/rng.
c   C1 = Coefficient used in vertical polarization calculations.
c   C2 = Coefficient used in vertical polarization calculations.
c   C1M = Constant for each calculated ALPHAV - used in C1 calculation.
c   C2M = Constant for each calculated ALPHAV - used in C2 calculation.
c   IG = Counter indicating current ground type being modeled.
c   RAV() = array of ROOT to the i'th power, i.e. RAV(I) = ROOT**I
c   RK = Coefficient used in C1 and C2 calculations.
c   RNG = complex refractive index.
c   RNG2 = complex refractive index squared.
c   ROOT = complex root of quadratic equation for mixed transform method
c          based on Kuttler's formulation.

c MISCVAR:
c   ANTREF = transmitting antenna height relative to the reference
c            height HMINTER.
c   CNST = used in calculating ENVPR() in routine PHASE1.
c          CNST = DELP/FKO.
c   DELP = mesh size in angle- (or p-) space.
c   FNORM = normalization factor used for DFT.
c   FTER = logical flag - .TRUE.=terrain case, .FALSE.=smooth surface case
c   HLIM() = array containing height at each output range at which the
c            last valid loss value exists.
c   HMREF = height relative to HMINTER.  Determined from user-provided
c           minimum height VNP.HMIN. That is,
c           if VNP.HMIN is minimum height input by user with respect to
c           mean sea level, and HMINTER is internally considered the new
c           origin, then HMREF = VNP.HMIN - HMINTER.
c   PLCNST = constant used in determining propagation loss
c            PLCNST = 20log(2*FKO).
c   QI = imaginary i -> complex(0,1)
c   RPE = range at which valid loss values will begin to be calculated.
c   THETAMAX = maximum propagation angle in PE calculations.
c   SLP() = slope of each segment of terrain.


c PARINIT:
c   HT() = height array of size N
c   HTDUM() = dummy array containing height values for current (interpolated)
c             profile. When using BS method this represents height values with
c             respect to the local ground height.
c   IS = counter for current profile (for range-dependent cases)
c   LVLEP = Number of height/refractivity levels in profile.
c   PROFINT() = M-unit profile interpolated to every DELZ in height
c   REFDUM() = dummy array containing M-unit values for current (interpolated)
c              profile. When using BS method this represents refractivity
c              profile with respect to the local ground height.
c   RV2 = range of the next refractivity profile (for range-dependent cases)

c PATTERN:
c   AFAC = constant used in determining antenna pattern factors
```

```
c          AFAC = 1.39157 / sin( bw / 2 ) for SIN(X)/X and height-finder
c          AFAC = (.5*ln(2))/(sin(bw/2))**2 for GAUSSIAN
c   BW = antenna pattern beamwidth in radians
c   ELV = antenna pattern elevation angle in radians
c   PELEV = sine of elevation angle
c   SBW = sine of the beamwidth
c   UMAX = limiting angle used in 30 dB cut-off point for SIN(X)/X and
c          generic height-finder antenna pattern factors

c PEVAR:
c   CON = 1.e-6 * FKO; Constant used in calculation of ENVPR()
c   DELZ = Bin width in z-space = WL / (2*sin(THETAMAX))
c   DZ2 = 2. * DELZ
c   FKO = free-space wavenumber = WL / (2*pi)
c   LN = Power of 2 transform size, i.e. N = 2**LN
c   N = Transform size
c   N34 = 3/4 * N
c   NM1 = N-1
c   WL = Wavelength in meters
c   ZMAX = Maximum height of PE calculation domain = N * DELZ

c RHSTPS:
c   DR = PE range step in meters
c   DR2 = 1/2 PE range step in meters
c   DROUT = Output range step in meters
c   DZOUT = Output height increment in meters
c   ZOUT() = array containing all output height points


c TRVAR:
c   DMDH() = gradients of first profile in M-units/meters
c   JLS = index of refractivity array at which antenna height is located
c   RLIM = 90% of maximum range RMAX - used for ray tracing
c   THETALAUNCH = angle in radians of launch angle for which, when traced,
c                 height of the ray at each output range step is stored.
c   ZLIM = height limit for ray trace

      common / arrays / u(0:maxpts), filt(0:maxn4), frsp(0:maxpts),
     +                  envpr(0:maxpts), ulst(0:maxpts)
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)
      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                   antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                   hmref
      common / pattern / pelev, afac, bw, elv, umax, sbw
      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep
      common / htvar / ylast, ycur, ycurm
      common / trvar / dmdh(mxlvls), zlim, jls, thetalaunch, rlim
      common / impedance / alphav, rav(0:maxpts), rng, rng2, c1, c2,
     +                     rk, c1m, c2m, ig, root

      record / errorflag / ef
      record / inputvar / vnp
      record / refractivity / rf
      record / systemvar / sv
      record / terrain / tr

      complex u, frsp, envpr, ulst, qi, alphav, rav, rng, rng2, c1c
      complex c2c, c2m, rk, c1m, c1, c2, root, ui, unmi, cx

      logical fter, lopen, lflag

      data radc / 1.74533e-2 /      !degree to radian conversion factor
      data iscr / 20 /                 ! Unit number for scratch file
      data c0 / 299.79245 /         !speed of light x 1e.-6 m/s
      data sdeg10 / .173648177 /       ! Sine of 10 degrees
```

```
      data sdeg15 / .258819045 /       ! Sine of 15 degrees
      data hdeg / 8.726646e-3 /      ! 1/2 degree

      ierror = 0
      fter = .false.
      thetamax = 0.
      hminter = 0.
      angu = 0.
      antref = sv.antht

c Put lower limit on HMAX and RMAX

      vnp.rmax = amax1( vnp.rmax, 5000. ) !Set max. range to no less than 5
                                          !km.
      vnp.hmax = amax1( vnp.hmax, 100. )  !Set max. height to no less than 100
                                          !m.
      if( vnp.hmin .ge. vnp.hmax ) then
         ierror = -42
         return
      end if
      vnp.hmin = amin1( vnp.hmin, vnp.hmax-100. )

      dzout = (vnp.hmax-vnp.hmin) / float( vnp.nzout )
      drout = vnp.rmax / float( vnp.nrout )

c Setup output height array

      do i = 1, vnp.nzout
         zout(i) = vnp.hmin + float(i) * dzout
      end do

      WL = c0 / sv.freq
      FKo = 2. * pi / WL
      con = 1.e-6 * fko
      qi = cmplx( 0., 1. )

c  Calculate constants used to determine antenna pattern factor
c SV.IPAT = 0 -> omni
c SV.IPAT = 1 -> gaussian
c SV.IPAT = 2 -> sinc x
c SV.IPAT = 3 -> csc**2 x
c SV.IPAT = 4 -> generic height-finder

      sv.bwidth = amax1( sv.bwidth, .5 )    !Don't let vertical beamwidth go
      sv.bwidth = amin1( sv.bwidth, 45. )   !outside .5 to 45 degree limit.

      sv.elev = amax1( sv.elev, -10. ) !Don't let elevation angle go
      sv.elev = amin1( sv.elev, 10. )  !outside -10 to 10 degree limit.

      bw = sv.bwidth * radc
      elv = sv.elev * radc
      bw2 = .5 * bw
      if( sv.ipat .eq. 1 ) then
         afac = .34657359 / (sin( bw2 ))**2
         pelev = sin( elv )
      elseif( sv.ipat .eq. 3 ) then
         sbw = sin( bw )
      elseif( sv.ipat .ne. 0 ) then
         afac = 1.39157 / sin( bw2 )
         a = pi / afac
         umax = atan( a / sqrt(1. - a**2)  )
      end if

c Discard any unnecessary terrain points.  Test on the rate of change of
c slope,i.e., second derivative.  If that is below 1.e-3 then discard that
c point.
```

```
      if( tr.itp .gt. 0 ) fter = .true.
      if( fter ) then

c Check that all terrain range points are increasing.

         do i = 1, tr.itp-1
            ip1 = i + 1
            if( tr.terx(ip1) .lt. tr.terx(i) ) then
               ierror = -17
               return
            end if
         end do

c Test to see that first range value is 0.

         if( tr.terx(1) .gt. 0. ) then
            ierror = -18
            return
         end if

c Test to see if the last range point in the profile meets or exceeds RMAX.
c If not then return error code.

         if( tr.terx(tr.itp) .lt. vnp.rmax ) then
            if( ef.lerr6 ) then
               ierror = -6
               return
            else
               tr.itp = min(tr.itp + 1, mxter)
               tr.terx(tr.itp) = vnp.rmax*1.01
               tr.tery(tr.itp) = tr.tery(tr.itp - 1)
            end if
         end if

c Test to see if the unit number for the scratch file is already attached to
c another file.  If so, search for a unit number that is unattached.

         inquire( iscr, OPENED = lopen )
         do while (lopen)
            iscr = iscr + 1
            inquire( iscr, OPENED = lopen )
         end do

         open( iscr, status = 'SCRATCH')
         write(iscr,*) tr.terx(1), tr.tery(1)      ! Keep first point of
         do i = 2, tr.itp-1                        ! terrain profile.
            im1 = i - 1
            ip1 = i + 1
            xm1 = tr.terx(im1)
            ym1 = tr.tery(im1)
            xi = tr.terx(i)
            yi = tr.tery(i)
            xp1 = tr.terx(ip1)
            yp1 = tr.tery(ip1)

            dx1 = amax1( 1.e-3, xi - xm1 )
            dx2 = amax1( 1.e-3, xp1 - xi )

            sl1 = (yi - ym1) / dx1
            sl2 = (yp1 - yi) / dx2

            scd = sl2 - sl1          ! dx is taken to be 1 m

c If second derivative is large enough then keep this point.

            if( abs(scd) .GT. 1.e-3 ) write(iscr,*) xi, yi
```

```
         end do

         write(iscr,*) tr.terx(tr.itp), tr.tery(tr.itp)      !Keep last point
         rewind( iscr )                                       !in profile.

c Now the scratch file contains all the necessary points for the terrain
c profile. Go back and read them in the arrays TR.TERX, TR.TERY.

         bugfix = 0.
         lflag = eof(iscr)
         tr.itp = 0
         do while( .not. lflag )
            tr.itp = tr.itp + 1
            read(iscr,*) tr.terx(tr.itp), tr.tery(tr.itp)
            if( tr.terx(tr.itp) .ge. vnp.rmax ) exit
            bugfix = 0.
            lflag = eof(iscr)
         end do

         close(iscr)

c Determine minimum height of terrain profile.  Then adjust entire terrain
c profile by this minimum height HMINTER such that this is the new 0
c reference.

         hminter = vnp.hmax
         do i = 1, tr.itp
            yi = tr.tery(i)
            if( yi .lt. hminter ) hminter = yi
         end do

c Get maximum height of terrain, then adjust terrain elevations by height
c offset.

         htermax = 0.
         do i = 1, tr.itp
            if( tr.tery(i) .gt. htermax ) htermax = tr.tery(i)
            tr.tery(i) = tr.tery(i) - hminter
         end do

c Return error code if VNP.HMAX does not exceed the maximum height of the
c terrain profile.

         if( htermax .gt. vnp.hmax ) then
            ierror = -8
            return
         end if

         antref = sv.antht + tr.tery(1)

         do i = 1, tr.itp-1

            ip1 = i + 1
            y1 = tr.tery(i)
            x1 = tr.terx(i)
            y2 = tr.tery(ip1)
            x2 = tr.terx(ip1)

            xdif = x2 - x1
            ydif = y2 - y1
            xdif = amax1( xdif, 1.e-5 )
            slope = ydif / xdif

            slp(i) = slope

c Calculate angle made from each terrain point height to antenna height above
c reference (HMINTER).  Determine maximum propagation angle so that direct ray
```

```
c angle will clear highest peak.

            if( y1 .gt. antref ) then
               angle = atan( (y1-antref) / x1 )
               if( angle .gt. angu ) angu = angle
            end if

         end do

c  Add 1/2 degree to the angle that clears the highest peak.

         angu = angu + hdeg

      end if

      hmref = vnp.hmin - hminter
      zlim = amax1( vnp.hmax-hminter, antref )

c Initialize refractivity arrays.

      call refinit( ef.lerr12, vnp.rmax, rf, IERROR )
      if( ierror .ne. 0 ) return

c Calculate gradients and other variables for use in ray tracing.

      do i = 1, lvlep-1
         rm1 = refdum(i)
         rm2 = refdum(i+1)
         h1 = htdum(i)
         h2 = htdum(i+1)
         grad = ( rm2 - rm1 ) / ( h2 - h1 )
         if( abs( grad ) .lt. 1.e-3 ) grad = sign( 1., grad )*1.e-3
         dmdh(i) = grad * 1.e-6      ! for ray trace formulas
      end do

      jls = 0
      rmatht = 0.
      do i = 1, lvlep-1
         if((antref .lt. htdum(i+1)).and.(antref .ge. htdum(i))) then
            jls = i
            rmatht = refdum(i) + (antref - htdum(i)) * dmdh(i)*1.e6
            exit
         end if
      end do

      rlim = .9 * vnp.rmax
      prang = vnp.propang * radc

c Calculate the critical angle and perform ray trace to get the maximum
c propagation angle such that you get coverage at all heights and ranges
c >= 90% of maximum range.  This is done for automatic angle calculation.

c Get minimum M-unit value of profile for all heights above transmitter
c height.

      j = jls + 1
      rmina = refdum(j)
      do i = j, lvlep
         if( refdum(i) .lt. rmina ) rmina = refdum(i)
      end do

c Get minimum M-unit value of profile for all heights below transmitter
c height.

      rminb = refdum(jls)
      do i = jls, 1, -1
         if( refdum(i) .lt. rminb ) rminb = refdum(i)
```

```
      end do

c Get critical angle if the transmitter is within or above a duct.

      acrit = 0.
      acrit1 = 0.
      acrit2 = 0.
      delm1 = rmatht - rmina
      delm2 = rmatht - rminb
      if( delm1 .gt. 0. ) acrit1 = sqrt( 2.e-6 * delm1 )
      if( delm2 .gt. 0. ) acrit2 = sqrt( 2.e-6 * delm2 )
      acrit = amax1( acrit1, acrit2 ) + 1.e-4

      thetamax = acrit

      at = atan( (zlim-antref) / vnp.rmax )
      thetamax = amax1( angu, at, acrit )

c If user inputs non-zero propagation angle, use that value.

      if( prang .ge. 1.e-6 ) thetamax = prang

c Get THETAMAX based on shallowest reflected ray traced to reach maximum
c height and still be within 90% of maximum range (for smooth surface). For
c terrain case the direct ray angle is used.

      call tracea( prang, acrit )

c Add buffer for filter region.

      thetamax = thetamax / .75

c Put lower limit on THETAMAX depending on frequency ( in MHz ):
c for 5000 <= SV.FREQ <= 9000,  THETAMAX >= .5 deg
c for 4100 <= SV.FREQ < 5000,  THETAMAX >= .6 deg
c for 2900 <= SV.FREQ < 4100,  THETAMAX >= .7 deg
c for 2500 <= SV.FREQ < 2900,  THETAMAX >= .8 deg
c for 1500 <= SV.FREQ < 2500,  THETAMAX >= .9 deg
c for 600 < SV.FREQ < 1500,  THETAMAX >= 1 deg
c for 400 < SV.FREQ <= 600,  THETAMAX >= 2 deg
c for 200 < SV.FREQ <= 400,  THETAMAX >= 3 deg
c for SV.FREQ <= 200,  THETAMAX >= 4 deg

      if( sv.freq .le. 9000. ) thetamax = amax1(thetamax, 8.72665e-3)
      if( sv.freq .lt. 5000. ) thetamax = amax1(thetamax, 1.047197e-2)
      if( sv.freq .lt. 4100. ) thetamax = amax1(thetamax, 1.22173e-2)
      if( sv.freq .lt. 2900. ) thetamax = amax1(thetamax, 1.396263e-2)
      if( sv.freq .lt. 2500. ) thetamax = amax1(thetamax, 1.570796e-2)
      if( sv.freq .lt. 1500. ) thetamax = amax1(thetamax, 1.745329e-2)
      if( sv.freq .le. 600. ) thetamax = amax1(thetamax, 3.4906585e-2)
      if( sv.freq .le. 400. ) thetamax = amax1(thetamax, 5.2359877e-2)
      if( sv.freq .le. 200. ) thetamax = amax1(thetamax, 6.981317e-2)

      if(( sv.polar .eq. 'V' ) .and. ( sv.freq .le. 500. ) .and.
     + ( prang .le. 1.e-6 )) thetamax = 2. * thetamax

c Get FFT size based on THETAMAX

      call getfftsz( ZLIM, IERROR )

c Maximize THETAMAX within determined FFT size for terrain cases and if
c using automatic angle calculation.

      if(( fter ) .and. (prang .le. 1.e-6)) then

c Use 74% of ZMAX instead of 75% to leave some slop and ensure the FFT size is
c not surpassed.
```

```
         if( .74*zmax .gt. zlim ) then
            thetafrac = thetalaunch / thetamax
            zmax = zlim / .74
            sthetamax = float(n) * wl * .5 / zmax

c Put upper limits on THETAMAX depending on frequency.

            if( sv.freq .gt. 1000. ) then
                sthetamax = amin1( sthetamax, sdeg10 )
            else
                sthetamax = amin1( sthetamax, sdeg15 )
            end if
            delz= wl * .5 / sthetamax
            thetamax = asin( sthetamax )
            zmax = float(n) * delz
            thetalaunch = thetafrac * thetamax
         end if
      end if

c Determine horizon range based on transmitter height and 0 receiver height
c by RHOR = 3572. * sqrt( 1.3333 * antref )

      rhor = 4124.5387 * sqrt( sv.antht )
      dr = 2. * fko * delz**2
      rkm = vnp.rmax * 1.e-3

c Determine range step.

      if( fter ) then
         dr = amin1( dr, 700. )
         if( rkm .ge. 5. ) rllim = 75.
         if( rkm .ge. 10. ) rllim = 90.
         if( rkm .ge. 15. ) rllim = 100.
         if( rkm .ge. 20. ) rllim = 110.
         if( rkm .ge. 30. ) rllim = 175.
         if( rkm .ge. 50. ) rllim = 200.
         if( rkm .ge. 75. ) rllim = 250.
         if( rkm .ge. 100. ) rllim = 300.
         dr = amax1( dr, rllim )
      else
         dr = amin1( dr, 1000. )
         dr = amax1( dr, 30. )
         if( vnp.rmax .ge. rhor ) dr = amax1( 300., dr )
      end if
      dr2 = .5 * dr

c path loss constant, add to TL to get PL:

      plcnst=20.*alog10(2.*fko)

c Initialize variables for free-space propagator phase calculations.

      delp = pi/zmax
      FNorm = 2. / N
      cnst = delp / fko
      nm1 = n - 1
      dz2 = 2. * delz

c Initialize variables and set-up filter array.

      no4 = n/4
      n34 = 3.* no4
      cn75 = 4.* pi / N
      do i = 0, no4
         fj= cn75 * float(i)
         filt(i) = .5 + .5 * cos(fj)
```

```
      end do

c Initialize dielectric ground constants for vertical polarization.

      ig = 1
      if( tr.igr .eq. 0 ) then
         tr.igr = 1
         tr.rgrnd(1) = 0.
         tr.igrnd(1) = 0
      end if
      if( sv.polar .eq. 'V' ) call dieinit( sv, tr )

c Initialize starter field.

      call xyinit( sv, tr )

c Transform to z-space.

      call fft( u )

c Initialize C1 and C2 to for start of PE calculations - only for vertical
c polarization.  C1 and C2 calculations only necessary at frequencies below
c 500 MHz.  NOTE: THIS IS ONLY FOR SMOOTH SURFACE.

      if(( sv.polar .eq. 'V' ) .and. ( sv.freq .le. 500. )) then
         c1c = cmplx( 0., 0. )
         c2c = cmplx( 0., 0. )
         do i = 0, n
            nmi = n - i
            ui = u(i)
            unmi = u(nmi)
            if(( i .eq. 0 ) .or. (i .eq. n )) then
               ui = .5 * ui
               unmi = .5 * unmi
            end if

            iv = mod( i, 2 )
            cx = rav(i)
            if( iv .eq. 1 ) cx = -rav(i)
            c1c = ui * rav(i)
            c2c = unmi * cx

            c1 = c1 + c1c
            c2 = c2 + c2c
         end do
         c1 = c1 * rk
         c2 = c2 * rk

      end if

      ylast = 0.
      if( fter ) ylast = tr.tery(1)

      ycurm = 0.
      rout = 0.
      ycur = 0.

c Define mesh array in height

      do i=0,n
         ht(i)= float(i)*delz
      end do

c If smooth surface, trace THETALAUNCH ray and store all heights at each
c output range step in array HLIM().

      call traceh( vnp.nrout )
```

```
c Determine the free-space propagator (p-space) arrays.

      call phase1

c If smooth surface and range-independent case then initialize all
c refractivity and z-space propagator arrays now.

      if(( .not. fter ) .and. (rf.nprof .eq. 1 )) then
         call profref( hminter, 0 )
         call intprof
         call phase2
      end if

      end
c *************************** SUBROUTINE PESTEP ******************************

c Purpose:  Propagates the field by one output range step DROUT.

c Called from MAIN DRIVER PROGRAM

c Routines called: DOSHIFT, SINFFT, REFINTER, PHASE2, CALCLOS

c PARAMETER LIST:

c Parameters passed:
c HMINTER = Minimum height of user-provided terrain profile.  This is
c           the height for which all internal calculations of the field
c           are referenced.
c VNP = structure of user input variables from input file.
c RF = structure of user-provided refractivity profiles.
c TR = structure of user-provided terrain data.
c SV = structure of user-provided radar system information.

c Parameters returned.
c JEND = index at which the valid propagation loss values end.
c JSTART = index at which the valid propagation loss values begin.
c MLOSS() = Array containing the propagation loss values in centibels,
c           at each output range point ROUT. All loss values returned
c           are referenced to height VNP.HMIN.
c ROUT = output range in meters.

      subroutine pestep( hminter, vnp, rf, tr, sv, ROUT, MLOSS, JSTART,
     +                   JEND )

      include 'tpem.inc'

      common / htvar / ylast, ycur, ycurm
      common / arrays / u(0:maxpts), filt(0:maxn4), frsp(0:maxpts),
     +                  envpr(0:maxpts), ulst(0:maxpts)
      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                   antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                   hmref
      common / impedance / alphav, rav(0:maxpts), rng, rng2, c1, c2,
     +                     rk, c1m, c2m, ig, root

      record / inputvar / vnp
      record / refractivity / rf
      record / terrain / tr
      record / systemvar / sv

      logical fter

      complex c1, c2, rk, c1m, c2m, qi, ar, br, sum1, sum2, c1c, c2c
      complex alphav, rav, rng, rng2, ui, unmi, root, arx, brx, cd
```

```
      complex u, frsp, envpr, ulst, w(0:maxpts), ym(0:maxpts)

      integer*2 mloss(*)

      save r, kt, slope

      if( rout .le. 1.e-3 ) r = 0.
      rout = rout + drout

      DO while( r .lt. rout )

         if( r .gt. 0. ) ylast = ycur
         rlast = r

c Store the field arrays of the previous range step for subsequent horizontal
c interpolation at range ROUT.

         do i = 0, n
            ulst(i) = u(i)
         end do

         r = r + dr
         rmid = r - dr2

         if( fter ) then
            if( abs(r - dr) .le. 1.e-3 ) then
               slope = slp(1)
               kt = 1
            end if

c Check to see if current range is past a range point in terrain profile.
c If so, increment counter, determine terrain height at current range.

            do while((r .gt. tr.terx(kt+1)) .and. (kt .lt. tr.itp))
               kt = kt + 1
               slope = slp(kt)
            end do
            ycur = tr.tery(kt) + slope * ( r - tr.terx(kt) )

c Determine height at 1/2 range step - for interpolation on refractivity
c profiles.

            kp = kt
            do while( rmid .lt. tr.terx(kp) )
               kp = kp-1
            end do
            ycurm = tr.tery(kp) + slp(kp) * (rmid - tr.terx(kp))

c Calculate new complex refractive index and impedance term if using vertical
c polarization.

            if( sv.polar .eq. 'V' ) then
               if( r .gt. tr.rgrnd(ig+1) ) then
                  ig=ig + 1
                  call getaln( tr )
               end if
            end if

c Perform boundary shift for terrain case.

            if( slope .lt. 0. ) call doshift
         end if

         if( sv.polar .eq. 'V' ) then
            do i = 1, nm1
               w(i) = (u(i+1) - u(i-1)) / dz2 + alphav * u(i)
            end do
```

```
            call frstp( frsp, W )

c Propagate C1 and C2 coefficients to new range. C1 and C2 calculations
c necessary only for frequencies below 500 MHz.  NOTE: ONLY FOR SMOOTH
c SURFACE.

            if( sv.freq .le. 500.) then
               c1 = c1 * c1m
               c2 = c2 * c2m
            end if
         else
            call frstp( frsp, U )
         end if

c If range-dependent and/or terrain case, then interpolate on profile.

         if(( rf.nprof .gt. 1 ) .or. ( fter )) then
           call refinter( rf, rmid, hminter )
           CALL PHASE2
         end if

c This follows steps 9-11 in Kuttler's formulation for vertical
c polarization. (Ref. viewgraphs from 1995 PE Workshop)

         if( sv.polar .eq. 'V' ) then
            ym(0) = cmplx(0.,0.)
            do i = 1, nm1
               ym(i) = dz2 * w(i) + root * ym(i-1)
            end do
            u(n) = cmplx(0.,0.)
            do i = 1, N
               nmi = n - i
               u(nmi) = root * (ym(nmi) - u(nmi+1))
            end do

c Calculations within loop necessary only for frequencies below 500 MHz.
c NOTE: ONLY FOR SMOOTH SURFACE.

            if( sv.freq .le. 500. ) then
               sum1 = cmplx( 0., 0. )
               sum2 = cmplx( 0., 0. )
               do i = 0, n
                  nmi = n - i
                  ui = u(i)
                  unmi = u(nmi)
                  if(( i .eq. 0 ) .or. (i .eq. n )) then
                     ui = .5 * ui
                     unmi = .5 * unmi
                  end if

                  iv = mod( i, 2 )
                  cd = rav(i)
                  if( iv .eq. 1 ) cd = -rav(i)

                  c1c = ui * rav(i)
                  c2c = unmi * cd

                  sum1 = sum1 + c1c
                  sum2 = sum2 + c2c
               end do

               ar = c1 - rk * sum1
               br = c2 - rk * sum2

               do i = 0, n
                  arx = ar * rav(i)
                  nmi = n - i
```

```
                 iv = mod( nmi, 2 )
                 cd = rav(nmi)
                 if( iv .eq. 1 ) cd = -rav(nmi)
                 brx = br * cd
                 u(i) = u(i) + arx + brx
              end do
           end if
        end if

c Multiply by environment term.

        DO I = 1, nml
           u(i) = u(i) * envpr(i)
        end do

c Perform boundary shift for terrain case.

        if(( fter ) .and. ( slope .ge. 0. )) call doshift

      end do

c Calculate propagation loss at range ROUT.

      call calclos( r, rout, rlast, vnp, hminter, MLOSS, JSTART, JEND )

      end

c ************************** SUBROUTINE ANTPAT ******************************

c Purpose: Determines the antenna pattern factor for angle passed to routine.

c Called from: XYINIT

c Routines called:  NONE

c PARAMETER LIST:

c Parameters passed:
c IPTRN = type of antenna pattern
c SANG = sine of angle

c Parameters returned:
c PATFAC = antenna pattern factor

      subroutine antpat( iptrn, sang, PATFAC )

      common / pattern / pelev, afac, bw, elv, umax, sbw

c In the following pattern definitions, "u" refers to the angle for which
c the antenna pattern is sought, and "u0" refers to the elevation angle.

c  IPTRN = 0 gives Omnidirectional antenna pattern factor : f(u) = 1

      patfac = 1.

      if( iptrn .gt. 1 ) then
         u = asin( sang )
         udif = u - elv
      end if

c  IPTRN = 1 gives Gaussian antenna pattern based on
c  f(p-p0) = exp(-w**2 * ( p-p0 )**2 ) / 4, where p = sin(u) and
c  p0 = sin(u0)

      if( iptrn .eq. 1 ) then
         pr = sang - pelev
         patfac = exp(-pr * pr * afac)
```

```
c IPTRN = 2 gives sin(x)/x pattern based on
c f(u-u0) = sin(x) / x where x = afac * sin(u-u0) for |u-u0| <= umax
c f(u-u0) = .03 for |u-u0| > umax
c IPTRN = 4 gives height-finder pattern which is a special case of sin(x)/x

      elseif(( iptrn .eq. 2 ) .or. ( iptrn .eq. 4 )) then
         if( iptrn .eq. 4 ) then
            dirang = abs( sang )
            if( dirang .gt. elv ) udif = u - dirang
         end if
         if( abs(udif) .le. 1.e-6 ) then
            patfac = 1.
         elseif( abs( udif ) .gt. umax ) then
            patfac = .03
         else
            arg = afac * sin( udif )
            patfac = amin1( 1., amax1( .03, sin( arg ) / arg ) )
         end if

c IPTRN = 3 gives csc-sq pattern based on
c f(u) = 1 for u-u0 <= bw
c f(u) = sin(bw) / sin(u-u0) for u-u0 > bw
c f(u) = maximum of .03 or [1+(u-u0)/bw] for u-u0 < 0

      elseif( iptrn .eq. 3 ) then
         if( udif .gt. bw ) then
            patfac = sbw / sin( udif )
         elseif( udif .lt. 0 ) then
            patfac = amin1( 1., amax1( .03, (1. + udif/bw) ) )
         end if
      end if

      end

c ***************************** SUBROUTINE CALCLOS **************************

c Purpose: Determines the propagation loss at each output range step ROUT and
c          all heights up to ZLIM.

c Called from: PESTEP

c Routines called: NONE
c Functions called: GETPFAC

c PARAMETER LIST:

c Parameters passed:
c   HMINTER = Reference height for internal calculations of the field U().
c   R = PE range step in meters
c   RLAST = last PE range in meters
c   ROUT = output range in meters
c   VNP = structure of user-provided input variables.


c Parameters returned:
c   JEND = index at which valid loss values in MLOSS ends.
c   JSTART = index at which valid loss values in MLOSS begin.
c   MLOSS() = 2 byte integer array containing propagation loss values in
c              centibels.

      subroutine calclos( r, rout, rlast, vnp, hminter, MLOSS, JSTART,
     +                    JEND )

      include 'tpem.inc'

      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
```

```
     +                    antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                    hmref
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)
      common / arrays / u(0:maxpts), filt(0:maxn4), frsp(0:maxpts),
     +                  envpr(0:maxpts), ulst(0:maxpts)
      common / htvar / ylast, ycur, ycurm
      common / trvar / dmdh(mxlvls), zlim, jls, thetalaunch, rlim

      record / inputvar / vnp

      complex u, frsp, envpr, ulst, qi

      integer*2 mloss(*)

      logical fter

      dimension rfac1(mxzout), rfac2(mxzout)

      save ic

      data pfacmin / 300. /       !Set minimum propagation factor of 300 dB

c Define in-line function for linear interpolation.

      plint(pl1, pl2, frac) = pl1 + frac * ( pl2 - pl1 )

c Initialize counter for HLIM array.

      if( abs(rout - drout) .le. 1.e-3 ) ic = 1

      ych = ycur - hmref
      yct = ycur + hminter
      ylh = ylast - hmref
      ylt = ylast + hminter

c Get height of ground at output range ROUT and determine number of vertical
c output points that correspond to the ground height.  Fill the loss array
c MLOSS with zeros to represent ground for those vertical output points.

      xx = (rout - rlast) / dr
      zint = plint( ylast, ycur, xx )
      izg = int( (zint-hmref) / dzout  )
      do i = 1, izg
         mloss(i) = 0
      end do

      jstart = amax0( 1, izg+1 )

      if( rout .gt. rpe ) then

c If current output range is greater than RPE then begin calculation of loss
c values and return them in MLOSS().

         rloglst = 0.
         if( rlast .gt. 0. ) rloglst = 10. * alog10( rlast )
         rlog = 10. * alog10( r )
         fslrout = 20. * alog10(rout) + plcnst       !free space loss at ROUT

c Determine values of array elements corresponding to the ground and set these
c to the minimum propagation factor (-300) for later interpolation.

         if( fter ) then
            ip1 = int( ylh / dzout )
            ip2 = int( ych / dzout )
            ip1 = amax0( 0, ip1 )
            ip2 = amax0( 0, ip2 )
```

```
            do i = 1, ip1
               rfac1(i) = pfacmin
            end do
            do i = 1, ip2
               rfac2(i) = pfacmin
            end do
            ip1 = ip1 + 1
            ip2 = ip2 + 1

         else
            ip1 = 1
            ip2 = 1
         end if

c Determine height/integer value at which to stop calculating loss.
c NOTE: For terrain cases, ray tracing was performed
c using the direct ray angle and sometimes HLIM(i) may be less than the
c local ground height. The GOTO statement is used just as a safety factor
c in this case.

         zend1 = amax1( zint, hlim(ic) )
         zend2 = amin1( zlim, zend1 )
         jend = amax0( 0, nint( (zend2-hmref) / dzout ) )

         if( jend .lt. jstart ) goto 5

c Get propagation factor at valid heights from field at previous range step.

         if( rloglst .gt. 0. ) then
            do i = ip1, jend
               zht = zout(i) - ylt
               rfac1(i) = getpfac( ulst, rloglst, zht )
            end do
         end if

c Get propagation factor at valid heights from field at current range step.

         do i = ip2, jend
            zht = zout(i) - yct
            rfac2(i) = getpfac( u, rlog, zht )
         end do

c Interpolate between the two PE range steps to get propagation loss at range
c ROUT.

         do k = jstart, jend
            if( rloglst .gt. 0. ) then
               loss = 10.*( plint( rfac1(k), rfac2(k), xx ) + fslrout )
               mloss(k) = int2( loss )
            else
               mloss(k) = int2( 10. * ( rfac2(k) + fslrout ) )
            end if
         end do

 5       continue

c Fill remainder of array with -1 indicating non-valid loss values.

         jn = jend + 1
         do i = jn, vnp.nzout
            mloss(i) = -1
         end do

      else

c If current output range is less than RPE then there are no current valid
```

```
c loss values at any height - fill MLOSS with -1. JSTART and JEND will be
c equal and will have a value of 1 if smooth surface case, otherwise will
c have a value of the nearest integer multiple of DZOUT corresponding to the
c height of the local ground.

         jend = jstart
         do i = jstart, vnp.nzout
            mloss(i) = -1
         end do

      end if

      ic = ic + 1

      end

c ************************ SUBROUTINE DIEINIT ****************************

c Purpose: This routine calculates Conductivity and Permittivity
c          as a function of frequency in MHz.  All equations and coef-
c          ficients were obtained by using a SUMMASKETCH digitizer with
c          the CCIR volume 5 curves on page 74.  The digitized data was
c          then used with the TABLECURVE software to obtain the best fit
c          equations and coefficients used in this subroutine.  In some
c          cases two sets of equations were required to obtain a decent
c          fit across the 100 MHz - 100GHz range.  These curves fit the
c          digitized data to within 5%.

c Called from: PEINIT

c Routines called: NONE

c PARAMETER LIST:

c Parameters passed:
c SV = structure of user-provided radar system inputs.
c TR = structure of user-provided terrain data.

      subroutine dieinit( sv, tr )

      include 'tpem.inc'

      record / terrain / tr
      record / systemvar / sv

      dimension a(14), b(14), c(14), d(14), e(14), f(14)

      data (a(i),i=1,14) / 1.4114535e-2, 3.8586749, 79.027635,
     +                     -0.65750351, 201.97103, 857.94335,
     +                     915.31026, 0.8756665, 5.5990969e-3,
     +                     215.87521, .17381269, 2.4625032e-2,
     +                     -4.9560275e-2, 2.2953743e-4 /
      data (b(i),i=1,14) / -5.2122497e-8, -2.1179295e-5, -2.2083308e-5,
     +                     5.5620223e-5, -2.5539582e-3, -8.9983662e-5,
     +                     -9.4530022e-6, 4.7236085e-5, 8.7798277e-5,
     +                     -7.6649237e-5, 1.2655183e-4, 1.8254018e-4,
     +                     2.9876572e-5, -8.1212741e-7 /
      data (c(i),i=1,14) / 5.8547829e-11, 9.1253873e-4, -3.5486605e-4,
     +                     6.6113198e-4, 1.2197967e-2, 5.5275278e-2,
     +                     -4.0348211e-3, 2.6051966e-8, 6.2451017e-8,
     +                     -2.6151055e-3, -1.6790756e-9, -2.664754e-8,
     +                     -3.0561848e-10, 1.8045461e-9 /
      data (d(i),i=1,14) / -7.6717423e-16, 6.5727504e-10, 2.7067836e-9,
     +                     3.0140816e-10, 3.7853169e-5, 8.8247139e-8,
     +                     4.892281e-8, -9.235936e-13, -7.1317207e-12,
     +                     1.2565999e-8, 1.1037608e-14, 7.6508732e-12,
     +                     1.1131828e-15, -1.960677e-12 /
```

```
      data (e(i),i=1,14) / 2.9856318e-21, 1.5309921e-8, 8.210184e-9,
     +                     1.4876952e-9, -1.728776e-6, 0.0,
     +                     7.4342897e-7, 1.4560078e-17, 4.2515914e-16,
     +                     1.9484482e-7, -2.9223433e-20, -7.4193268e-16,
     +                     0.0, 1.2569594e-15 /
      data (f(i),i=1,14) / 0., -1.9647664e-15, -1.0007669e-14, 0., 0.,
     +                     0., 0., -1.1129348e-22, -1.240806e-20, 0.,
     +                     0., 0., 0., -4.46811e-19 /

      f1 = sv.freq
      f2 = f1 * f1
      f3 = f1 * f2
      f4 = f1 * f3
      f5 = f1 * f4
      f6 = f1 * f5
      f7 = f1 * f6
      f8 = f1 * f7
      f9 = f1 * f8

c EPSILON = relative permittivity
c SIGMA = conductivity

      do i = 1, tr.igr

         select case ( tr.igrnd(i) )

            case( 0 )  ! Permittivity and conductivity for salt water
               epsilon = 70.
               sigma = 5.
               m = 1
               m1 = m + 1
               if( f1 .gt. 2253.5895 ) epsilon = 1. / ( a(m) +
     +            b(m)*f1 + c(m)*f2 + d(m)*f3 + e(m)*f4 )
               if( f1 .gt. 1106.207 ) then
                  sigma = a(m1) + c(m1)*f1 + e(m1)*f2
                  sigma = sigma / ( 1.+ b(m1)*f1 + d(m1)*f2 + f(m1)*f3 )
               end if

            case( 1 )  !Permittivity and conductivity for fresh water
               epsilon = 80.0
               m = 3
               m1 = m + 1
               IF( f1 .gt. 6165.776 ) THEN
                  epsilon = a(m) + c(m)*f1 + e(m)*f2
                  epsilon = epsilon/(1. + b(m)*f1 + d(m)*f2 + f(m)*f3 )
               end if
               IF( f1 .gt. 5776.157) THEN
                  k = 2
               else
                  m1 = m1 + 1
                  k = -1
               end if
               sigma = a(m1) + c(m1)*f1 + e(m1)*f2
               sigma = (sigma / (1. + b(m1)*f1 + d(m1)*f2))**k

            case( 2 ) !Permittivity and conductivity for wet ground
               epsilon = 30.0
               m = 6
               IF( f1 .ge. 4228.11 ) m = 7
               if( f1 .gt. 1312.054 ) then
                  epsilon = a(m) + c(m)*f1 + e(m)*f2
                  epsilon = SQRT( epsilon / (1. + b(m)*f1 + d(m)*f2) )
               end if
               IF( f1 .gt. 15454.4) then
                  m1 = 8
                  g = 3.3253339e-28
               else
```

```
                m1 = 9
                g = 1.3854354e-25
             end if
             sigma = a(m1) + b(m1)*f1 + c(m1)*f2 + d(m1)*f3 + e(m1)*f4
             sigma = sigma + f(m1)*f5 + g*f6

          case( 3 ) !Permittivity and conductivity for medium dry ground
             epsilon = 15.0
             IF( f1 .gt. 4841.945) THEN
                m = 10
                epsilon = a(m) + c(m)*f1 + e(m)*f2
                epsilon = SQRT( epsilon / (1. + b(m)*f1 + d(m)*f2) )
             end if
             m1 = 12
             IF( f1 .gt. 4946.751) m1 = 11
             sigma = (a(m1) + b(m1)*f1 + c(m1)*f2 + d(m1)*f3 +
     +                e(m1)*f4)**2

          case( 4 ) !Permittivity and conductivity for very dry ground
             epsilon = 3.0
             IF( f1 .lt. 590.8924 ) then
                sigma = 1.0e-4
             else
                IF( f1 .gt. 7131.933) THEN
                   m1 = 13
                   sigma = (a(m1) + b(m1)*f1 + c(m1)*f2 + d(m1)*f3)**2
                else
                   m1 = 14
                   g = 9.4623158e-23
                   h = -1.1787443e-26
                   s = 7.9254217e-31
                   t = -2.2088286e-35
                   sigma = a(m1) + b(m1)*f1 + c(m1)*f2 + d(m1)*f3
                   sigma = sigma + e(m1)*f4 + f(m1)*f5 + g*f6
                   sigma = sigma + h*f7 + s*f8 + t*f9
                end if
             end if

          case( 5 )
             epsilon = tr.dielec(1,i)
             sigma = tr.dielec(2,i)

          case default
             ! Do nothing
       end select

       tr.dielec(1,i) = epsilon
       tr.dielec(2,i) = sigma
    end do

c Set dielectric constants equal to last provided ground constants at 1e7 km.

    tr.igr = tr.igr + 1
    tr.rgrnd(tr.igr) = 1.e10
    tr.dielec(1,tr.igr) = epsilon
    tr.dielec(2,tr.igr) = sigma

    end

c ************************* SUBROUTINE DOSHIFT ****************************

c Purpose: Shifts the field by the # of bins corresponding to height of
c          the ground.

c Called from: PESTEP

c Routines called: NONE
```

```
c PARAMETER LIST:

c Parameters passed: NONE

c Parameters returned: NONE

      subroutine doshift

      include 'tpem.inc'

      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / htvar / ylast, ycur, ycurm
      common / arrays / u(0:maxpts), filt(0:maxn4), frsp(0:maxpts),
     +                  envpr(0:maxpts), ulst(0:maxpts)

      complex u, frsp, envpr, ulst

c Determine # of bins to shift field.

      ydif = ycur - ylast
      kbin = nint( abs(ydif) / delz )
      if( kbin .eq. 0 ) return

c If slope is positive then shift array elements down.

      if( ydif .ge. 0. ) then
         incr = 1
         jst = 1
         jend = nm1 - kbin
      else

c If slope is negative then shift array elements up.

         incr = -1
         jst = nm1
         jend = kbin + 1
      end if

      kinc = incr * kbin
      do j = jst, jend, incr
         jk = j + kinc
         u(j) = u(jk)
      end do

      if( incr .gt. 0 ) then
         nst = n - kbin
         do j = nst, nm1
            u(j) = 0.
         end do
      else
         do j = 1, kbin
            u(j) = 0.
         end do
      end if

      end
c *************************** SUBROUTINE FFT ********************************

c Purpose: Performs fast Fourier sine transform on complex array U.

c Called from: PEINIT, PESTEP

c Routines called: SINFFT

c PARAMETER LIST:
```

```
c Parameters passed:
c U = Complex field to be transformed.

c Paramters returned:
c U = Transform of complex field.

      subroutine fft( u )

      include 'tpem.inc'

      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1

      complex u(0:*)

      dimension x(0:maxpts), y(0:maxpts)

      do i = 0, n
         x(i) = real( u(i) )
         y(i) = imag( u(i) )
      end do

      call sinfft( ln, X )
      call sinfft( ln, Y )

      do i = 0, n
         u(i) = cmplx( x(i), y(i) )
      end do

      end

c ************************** SUBROUTINE FRSTP ******************************

c Purpose:  Propagates the field FARRAY() in free space by one range step.
c           If polarization is horizontal, then upon entry FARRAY() is the
c           field array U().  If using vertical polarization, FARRAY() is W().

c Called from: PESTEP

c Routines called: FFT

c PARAMETER LIST:

c Parameters passed:
c FRSP() = Complex free space propagator term.
c FARRAY() = Field array to be propagated one range step in free space
c             (z-space).

c Paramters returned:
c FARRAY() = Propagated field (returned in z-space).

      subroutine frstp( frsp, FARRAY )

      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1

      complex frsp(0:*), farray(0:*)

      call fft( farray )     !Transform to Fourier space

      DO I = 1, NM1                 !Multiply by free-space propagator
         farray(i) = farray(i) * frsp(i)
      end do

      call fft( farray )     !Transform back to z-space

      end
```

```
c ************************** SUBROUTINE GETALN ******************************

c Purpose:  Computes the impedance term ALPHAV and the complex index of
c           refraction for finite conductivity and vertical polarization
c           calculations. These formulas follow Kuttler's method. (Ref.
c           Kuttler's viewgraphs from PE modeler's workshop).

c Called from: XYINIT

c Routines called: NONE

c PARAMETERS LIST:

c Parameters passed:
c TR = structure of user-provided terrain data.

c Parameters returned: NONE

      subroutine getaln( tr )

      include 'tpem.inc'

      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / impedance / alphav, rav(0:maxpts), rng, rng2, c1, c2, rk,
     +                     c1m, c2m, ig, root
      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                   antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                   hmref

      record / terrain / tr

      complex alphav, rav, rng, rng2, c1, c2, rk, c1m, c2m, root
      complex rav1, rav2, ad, sqrad, r2, a, ravln, qi, r2n

      logical fter

      s1 = tr.dielec(2,ig) * 60. * wl
      rng2 = cmplx( tr.dielec(1,ig), s1 )
      rng = csqrt( rng2 )
      alphav = qi * fko / rng
      ad = alphav * delz

      sqrad = csqrt( 1. + ad**2 )

      rav1 = sqrad - ad
      rav2 = -sqrad - ad
      ravmag1 = cabs( rav1 )
      ravmag2 = cabs( rav2 )
      root = rav2
      if( ravmag1 .lt. ravmag2 ) root = rav1
      do i = 0, n
         rav(i) = root**i
      end do

      r2 = rav(2)
      r2n = rav(n)**2
      rk = 2.*(1. - r2) / (1. + r2) / (1. - r2n)
      a = dr * qi / 2. / fko
      ravln = clog( root )
      ad = (ravln / delz)**2
      c1m = cexp( a * ad )
      ad = ( (ravln - qi * pi ) / delz )**2
      c2m = cexp( a * ad )

      end
```

```
c ************************** SUBROUTINE GETFFTSZ ***************************

c Purpose:  Determines the FFT size needed for a given problem.

c Called from: PESTEP

c Routines called: NONE

c PARAMETER LIST:

c Parameters passed:
c ZLIM = Maximum height region where PE solution is valid = .75 * ZMAX.

c Parameters returned:
c ZLIM = Calculates a new height limit equal to .75*ZMAX only if the
c        maximum transform size needed is too large to do specified
c        problem.  Fixes transform size to maximum and adjusts ZMAX and
c        ZLIM accordingly.

      subroutine getfftsz( zlim, IERROR )

      include 'tpem.inc'

      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                   antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                   hmref
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1

      logical fter

      complex qi

      ierror = 0
      sthetamax = sin( thetamax )
      delz= wl * .5 / sthetamax

c Set lower FFT limit to 2**9 for smooth surface cases, if terrain case then
c set lower FFT limit to 2**10.

      ln = 9
      if( fter ) ln=10
      N=2**LN

      zmax=delz*float(n)

c Determine transform size needed to perform calculations to a height of ZLIM,
c up to the maximum FFT size allowed.

      do while( .75*zmax .lt. zlim )
         ln = ln + 1
         if( ln .gt. mxnfft ) exit
         n = 2**ln
         zmax = delz * float(n)
      end do

c If the transform size needed is too large then set ZMAX and ZLIM
c accordingly.

      if( ln .gt. mxnfft ) then
         ln = mxnfft
         n = 2**ln
         zmax = delz * float(n)
         zlim = .75 * zmax
      end if

      end
```

```
c **************************** FUNCTION GETPFAC ***************************

c Purpose:  Performs linear interpolation in height on the power and then
c           calculates propagation factor in dB.

c Called from: CALCLOS

c Routines called: NONE

c PARAMETER LIST:

c Parameters passed:
c HEIGHT = receiver height in meters
c RLOG = 10. * log( range )
c U = Complex field

c Parameters returned: GETPFAC

      function GETPFAC( u, rlog, height )

      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1

      complex u(0:*), u0, u1

      data powmin/1.e-13/

      fb = height / delz
      nb=int(fb)
      fr=fb-float(nb)
      nbp1=nb+1

      u0=u(nb)
      u1=u(nbp1)

      pow0 = cabs( u0 )
      pow1 = cabs( u1 )

      pow = pow0 + fr * (pow1 - pow0)

      rpow = amax1( pow, powmin )
      getpfac = -20.*alog10( rpow ) - rlog

      end

c ************************* SUBROUTINE INTPROF *****************************

c Purpose:  Performs a linear interpolation vertically with height on the
c           refractivity profile.  Stores interpolated profile in PROFINT().

c Called from: REFINTER, PESTEP

c Routines called: NONE

c PARAMETER LIST:

c Parameters passed: NONE

c Parameters returned: NONE

c Common blocks:

c PROFWREF:
c   HREF() = Heights of refractivity profile with respect to YREF.
c   NLVL = Number of levels in profile.
c   REFREF() = Refractivity array.

      SUBROUTINE intprof
```

```
      include 'tpem.inc'

      common / profwref / href(mxlvls), refref(mxlvls), nlvl
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                    profint(0:maxpts), ht(0:maxpts), is, lvlep

      J=2

      DO I=0,N
         height = ht(i)
   40    IF((height .LE. href(J)) .OR. (J .GE. nlvl)) then
            k = j - 1
            FRAC = (height - href(k)) / (href(J) - href(k))
            profint(I) = (refref(k) + FRAC * (refref(J) -
     +                    refref(k))) * con
         else
            J=J+1
            GO TO 40
         end if
      end do

      END

c ***************************** SUBROUTINE PHASE1 ****************************

c Purpose: Initialize free-space propagator array FRSP() using wide-angle
c          propagator.

c Called from: PEINIT

c Routines called: NONE

c PARAMETER LIST:

c Parameters passed: NONE

c Parameters returned: NONE

      SUBROUTINE PHASE1

      include 'tpem.inc'

      common / arrays / u(0:maxpts), filt(0:maxn4), frsp(0:maxpts),
     +                  envpr(0:maxpts), ulst(0:maxpts)
      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                   antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                   hmref
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)

      logical fter

      complex u, frsp, envpr, ulst, qi

      double precision cak

      drfk = dr * fko
      DO I=0,N
         ak = float(i) * cnst
         aksq=ak * ak
         aksq = amin1( 1., aksq )
         cak = sqrt(1. - aksq)
         ang = drfk * ( 1.d0 - cak )
         ca = cos( ang )
         sa = -sin( ang )
```

```
         frsp(i) = fnorm * cmplx( ca, sa )
      end do

c Filter the upper 1/4 of the propagator arrays.

      do  i = n34, n
         attn = filt(i-n34)
         frsp(i) = attn * frsp(i)
      end do

      END

c *************************** SUBROUTINE PHASE2 ******************************

c Purpose: Calculates the environmental phase term for a given profile, then
c          stores in array ENVPR().

c Called from: PEINIT, PESTEP

c Routines called: NONE

c PARAMETER LIST:

c Parameters passed: NONE

c Parameters returned: NONE

      SUBROUTINE PHASE2

      include 'tpem.inc'

      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)
      common / arrays / u(0:maxpts), filt(0:maxn4), frsp(0:maxpts),
     +                  envpr(0:maxpts), ulst(0:maxpts)
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep

      complex u, frsp, envpr, ulst

      do i = 0, n
         ang = dr * profint(i)
         ca = cos( ang )
         sa = sin( ang )
         envpr(i) = cmplx( ca, sa )
      end do

c Filter upper 1/4 of the arrays.

      do i = n34, n
         attn = filt(i-n34)
         envpr(i) = attn * envpr(i)
      end do

      END

c ****************************** SUBROUTINE PROFREF ************************

c Purpose: This subroutine determines the refractivity profile with respect
c          to the reference height YREF which, depending on the value of
c          IFLAG,can be HMINTER or the local ground height above HMINTER.

c Called from:  PEINIT, REFINTER

c Routines called: NONE

c PARAMETER LIST:
```

```
c Parameters passed:
c IFLAG = 0: Profile arrays REFREF() and HREF() will be referenced to height
c             HMINTER, and will also be used to initialize REFDUM() and
c             HTDUM().
        = 1: Profile arrays REFREF() and HREF() will be referenced to the
c             local ground height.
c YREF = Reference height in meters at current range step.

c Parameters returned:NONE

c Common blocks:

c PROFWREF:
c   HREF() = Heights of refractivity profile with respect to YREF.
c   NLVL = Number of levels in profile.
c   REFREF() = Refractivity array.

      subroutine profref( yref, iflag )

      include 'tpem.inc'

      common / profwref / href(mxlvls), refref(mxlvls), nlvl
      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep

      nlvl = lvlep
      if( yref .gt. 0 ) then

         do i = 1, mxlvls
            href(i) = 0.
            refref(i) = 0.
         end do

c Get refractivity profile level at which the height of the ground is just
c above. This level is JS.

         js = 0
         nlvlm1 = nlvl - 1
         do i = 1, nlvlm1
            if(( yref .le. htdum(i+1) ) .and. ( yref .gt. htdum(i) ))
     +         js = i
         end do

c Determine the refractivity value at the ground and fill arrays HREF() and
c REFREF() with refractivity profile where height 0. now refers to the ground
c reference,i.e., either local ground height or HMINTER.

         if( js .ne. 0 ) then
            jsp1 = js + 1
            frac = (yref - htdum(js))/(htdum(jsp1) - htdum(js))
            rmu = refdum(js) + frac * (refdum(jsp1) - refdum(js))
            if( int( frac ) .eq. 1 ) js = jsp1
            newl = nlvl - js +1
            refref(1) = rmu
            href(1) = 0.
            k = js + 1
            do jk = 2, newl
               refref(jk) = refdum(k)
               href(jk) = htdum(k) - yref
               k = k + 1
            end do
            nlvl = newl
            if( iflag .eq. 0 ) then
               lvlep = nlvl
               do i = 1, mxlvls
                  refdum(i) = refref(i)
```

```
                  htdum(i) = href(i)
               end do
            end if
         end if
      else

c If the reference height is 0. then HREF() and REFREF() are equal.

         do i = 1, nlvl
            href(i) = htdum(i)
            refref(i) = refdum(i)
         end do
      end if

      end

c ************************* SUBROUTINE REFINIT *****************************

c Purpose: Initializes refractivity arrays used for subsequent PE
c calculations.

c Called from: PEINIT

c Routines called: REMDUP

c PARAMETER LIST:

c Parameters passed:
c ELERR12 = structure of user-provided error flags that will trap on certain
c           errors if set to .TRUE.  Refer to user's manual.
c RF = structure of user-provided refractivity inputs.
c VRMAX = Maximum range in meters

c Parameters returned:
c IERROR =  -6 : Last range in terrain profile is less than RMAX. (Will only
c                return this error if error flag EF.LERR6 is set to .TRUE.).
c          -12 : Range of last refractivity profile entered (for range depen-
c                dent case) is less than VRMAX. (This is returned from subrou-
c                tine REFINIT).  Will only return this error if error flag
c                EF.LERR12 is set to .TRUE.).

      subroutine refinit( elerr12, vrmax, rf, IERROR )

      include 'tpem.inc'

      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep
      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)

      record / refractivity / rf

      logical elerr12

      data hlarge/ 1.e6 /
      data rlarge / 1.e10 /

      ierror = 0

c Test to see if last profile entered ( for range dependent case ) meets or
c exceeds VRMAX, otherwise, return error (unless error trapping is turned off
c - EF.LERR12 = .FALSE.).

      if( rf.nprof .gt. 1 ) then
         if(( rf.rngprof(rf.nprof) .lt. vrmax ) .and. ( elerr12 )) then
            ierror = -12
            return
         end if
```

```
      end if

c Add extra level to tabulated profiles with extrapolated gradient.  Test on
c HDIF greater than 0 for profiles that contain multiple height/M-unit values
c that are equal.

      rf.lvlep = rf.lvlep + 1
      do i = 1,rf.nprof
         hdif = 0.
         lvlm1 = rf.lvlep
         lvlm2 = rf.lvlep
         do while( hdif .le. 1.e-6 )
            lvlm1 = lvlm1 - 1
            lvlm2 = lvlm1 - 1
            hdif = rf.hmsl(lvlm1,i) - rf.hmsl(lvlm2,i)
         end do
         grad = (rf.refmsl(lvlm1,i)-rf.refmsl(lvlm2,i)) / hdif

c If last gradient in refractivity profile is negative then return error.

         if( grad .lt. 0 ) then
            ierror = -14
            return
         end if

         rf.hmsl(rf.lvlep, i) = hlarge
         rf.refmsl( rf.lvlep, i ) = (hlarge-rf.hmsl(lvlm1,i)) * grad +
     +                    rf.refmsl( lvlm1, i )
      end do

      is = 1
      rv2=rf.rngprof(is)

      do i = 1, rf.lvlep
         refdum(i) = rf.refmsl( i, is )
         htdum(i) = rf.hmsl( i, is )
      end do

      np = rf.nprof + 1
      rf.rngprof(np) = rlarge
      do i = 1, rf.lvlep
         npm1 = np - 1
         rf.hmsl( i, np ) = rf.hmsl( i, npm1 )
         rf.refmsl( i, np ) = rf.refmsl( i, npm1 )
      end do

      lvlep = rf.lvlep
      call remdup

      end

c ************************** SUBROUTINE REFINTER *****************************

c Purpose: Interpolates vertically and horizontally on the refractivity
c          profiles.

c Called from: PESTEP

c Routines called: REMDUP, PROFREF, INTPROF

c PARAMETER LIST:

c Parameters passed:
c RANGE = Range for profile interpolation.
c RF = Structure of user-provided refractivity inputs.
c HMINTER = Reference height in meters
```

```
c Parameters returned:NONE

c Common blocks:

c PROFWREF:
c   HREF() = Heights of refractivity profile with respect to YREF.
c   NLVL = Number of levels in profile.
c   REFREF() = Refractivity array.

      subroutine refinter( rf, range, hminter )

      include 'tpem.inc'

      common / profwref / href(mxlvls), refref(mxlvls), nlvl
      common / htvar / ylast, ycur, ycurm
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep

      record / refractivity / rf

      save j, rv1
      data j, rv1 / 0, 0. /

c One-line interpolation function

      pint( p1, p2 ) = p1 + fv * ( p2 - p1 )

      lvlep = rf.lvlep

c If there is a range-dependent refractivity profile then interpolate
c horizontally using the two surrounding profiles at range RANGE with all
c duplicate levels.

      if( rf.nprof .gt. 1 ) then
         IF( range .gt. rv2 ) then
            j = is
            IS=IS+1
            rv1=rv2
            rv2=rf.rngprof(IS)
         end if

         FV=(range-rv1)/(rv2-rv1)

         do i = 1, lvlep
            refdum(i) = pint( rf.refmsl(i,j), rf.refmsl(i,is) )
            htdum(i) = pint( rf.hmsl(i,j), rf.hmsl(i,is) )
         end do

c Now remove all duplicate levels with LVLEP now being the # of points in the
c profile at range RANGE.

         call remdup
         call profref( hminter, 0 )

c At this point REFDUM() and HTDUM(), also HREF() and REFREF(), are referenced
c to HMINTER.

      end if

c Using BS method must determine height and M-unit profiles relative to
c ground,where YCURM is now the height of the local ground above the reference
c ground, HMINTER.

      call profref( ycurm, 1 )

c Interpolate vertically with height.  PROFINT is now an N-point (N=2**NFFT)
```

```
c array containing the interpolated M-unit values for the refractivity at
c range RANGE.

      call intprof

      end

c ************************* SUBROUTINE REMDUP *****************************

c Purpose: Removes duplicate refractivity levels in profile.

c Called from: REFINIT, REFINTER

c Routines called:  NONE

c PARAMETER LIST:

c Parameters passed: NONE

c Parameters returned: NONE

      subroutine remdup

      include 'tpem.inc'

      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep

c  Remove all duplicate levels in interpolated profile

      i = 1
      do while( i .lt. lvlep )
         ht1 = htdum(i)
         ht2 = htdum(i+1)
         if( abs(ht1-ht2) .le. 1.e-3 ) then
            lvlep = lvlep - 1
            do j = i, lvlep
               jp1 = j + 1
               htdum(j) = htdum(jp1)
               refdum(j) = refdum(jp1)
            end do
            i = i - 1
         end if
         i = i + 1
      end do

      end

c *************************** SUBROUTINE TRACEA ****************************

c Purpose:  This routine performs a ray trace to determine the minimum angle
c           required (based on the reflected ray) to obtain a PE solution for
c           all heights up to ZLIM and all ranges beyond RLIM. THETAMAX
c           is then determined from this angle.  This is done only for smooth
c           surface and automatic angle calculation.  For terrian cases,
c           THETAMAX has already been set to the larger of the critical angle
c           (if a duct exists), the angle that clears the highest terrain
c            peak,and the tangent angle determined from HMAX and RMAX.

c           If PRANG is not equal to 0, then the user has overriden the
c           default calculation and THETAMAX is then determined based on
c           PRANG.  However a ray trace must still be done in order to
c           determine the initial launch angle such that the local angle of
c           the ray remains less than PRANG. The initial launch angle is used
c           in subroutine TRACEH.

c Called from: PEINIT
```

```
c Routines called: NONE

c PARAMETER LIST:

c Parameters passed:
c PRANG =  Problem angle (rad) - input by the user. Default is 0.
c ACRIT =  Critical angle.  Angle above which no rays are trapped.

c Parameters returned: NONE

      subroutine tracea( prang, acrit )

      include 'tpem.inc'

      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                   antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                   hmref
      common / trvar / dmdh(mxlvls), zlim, jls, thetalaunch, rlim
      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep

      complex qi

      logical fter, loop

      data deg15 / .2617994 /

c All heights and ranges are in meters, gradients are in M-unit/meter * 1.e-6
c and angles are in radians.

c Define in line ray trace functions:

      radal( a, b ) = a**2 + 2. * grad * b               !a=a0, b=h1-h0
      rp( a, b ) = a + b / grad                          !a=r0, b=a1-a0
      ap( a, b ) = a + b * grad                          !a=a0, b=r1-r0
      hp( a, b, c ) = a + ( b**2 - c**2 ) / 2. / grad    !a=h0, b=a1, c=a0

c AS = Starting launch angle in radians.
c ASL = Last (or previous) starting launch angle.
c AMXCUR = Maximum of local angle along ray.
c AMXCURL = Last (or previous) AMXCUR
c ISET = flag to test whether or not to stop loop

      as = -thetamax
      idn = -1
      if( fter ) then
         as = thetamax
         if( prang .le. 1.e-6 ) idn = 1
      end if
      asl = 0.
      amxcurl = 0.
      iset = 0

      do while( iset .eq. 0 )

c Decrease or increase angle by 1 mrad, depending on value of IDN.
c Initialize ray trace variables.

         as = as + idn*1.e-3
         h0 = antref
         r0 = 0.
         rpe = 0.
         a0 = as
         jl = jls
         amxcur = 0.
         loop = .true.
```

```
c Perform ray trace until ray has reached ZLIM and/or RLIM where
c ZLIM = maximum of HMAX-HMINTER or ANTREF.
c RLIM = .9 * RMAX

         do while( loop )

            grad = dmdh(jl)
            if( a0 .lt. 0. ) h1 = htdum(jl)
            if( a0 .gt. 0. ) h1 = htdum(jl + 1)
            if( a0 .eq. 0. ) then
               if( grad .lt. 0. ) h1 = htdum(jl)
               if( grad .gt. 0. ) h1 = htdum(jl+1)
            end if
            if( h1 .gt. zlim ) h1 = zlim
            rad = radal( a0, h1-h0 )
            if( rad .gt. 0 ) then
               a1 = sign( 1., a0 ) * sqrt( rad )
            else
               a1 = 0.
               h1 = hp( h0, a1, a0 )
            end if

            r1 = rp( r0, a1-a0 )

            if(( a1 .le. 0. ) .and. ( h1 .le. htdum(jl) )) then
               h1 = htdum(jl)
               rad = radal( a0, h1-h0 )
               a1 = -sqrt( rad )
               r1 = rp( r0, a1-a0 )
               jl = jl - 1
               if( jl .eq. 0 ) jl = 1
            elseif(( a1 .ge. 0. ).and.( h1 .ge. htdum(jl+1) ))then
               h1 = htdum(jl+1)
               rad = radal( a0, h1-h0 )
               a1 = sqrt( rad )
               r1 = rp( r0, a1-a0 )
               jl = jl + 1
               if( jl .gt. lvlep ) jl = lvlep
            end if

            if( h1 .gt. zlim ) then
               h1 = zlim
               rad = radal( a0, h1-h0 )
               a1 = sqrt( rad )
               r1 = rp( r0, a1-a0 )
            end if

            h0 = h1
            r0 = r1
            a0 = a1

c Set RPE to range at which reflected ray hits the ground.

            if( h0 .le. 1.e-4 ) then
               a0 = -a0
               rpe = r0
            end if

            if( a0 .ge. 1.57079 ) exit
            amxcur = amax1( amxcur, a0 )
            if(( h0 .ge. zlim ) .and. ( a0 .gt. 0.)) loop = .false.
            if( r0 .gt. rlim ) loop = .false.

         end do

c Test to see if the current ray traced from launch angle AS meets criteria.
```

```
c If ray traced does not reach ZLIM AND is not within RLIM then the initial
c launch angle AS is increased by 1 mrad and ray trace is
c repeated.  This is done only for smooth surface.

         if(( r0 .le. rlim ) .and. ( rpe .gt. 0. )) iset = 1

c If criteria is met then (if user specified a problem angle) make sure the
c local maximum angle is just within PRANG.  If not then repeat ray trace
c until this occurs.

         if( fter )then
            if( prang .gt. 1.e-6 ) then
               iset = 1
               if( amxcur .gt. thetamax ) iset = 0
               if( as .le. acrit+1.e-3 ) iset = 1   !Don't let launch angle
                                                    !be less than critical
                                                    !angle.
            else
               if(( r0 .le. rlim ) .and. ( h0 .ge. zlim )) iset = 1
               if( iset .eq. 1 ) thetamax = amax1( abs(as), amxcur )
            end if
         else
            if(( prang .gt. 1.e-6 ) .and. (iset .eq. 1)) then
               a = amax1( abs(as), amxcur )
               if( a .lt. prang ) then
                  iset = 0
               elseif( asl .ne. 0. ) then
                  as = asl
                  amxcur = amxcurl
               end if
            end if
         end if

c Just as a safeguard - set absolute maximum of launch angle to 15 degrees.

         if( as .le. -deg15 ) then
            iset = 1
            as = -deg15
            amxcur = deg15
         end if

         asl = as
         amxcurl = amxcur

      end do

      if( .not. fter ) thetamax = amax1( abs(as), amxcur )
      thetalaunch = abs(as)

      end

c **************************** SUBROUTINE TRACEH *****************************

c Purpose:  Computes ray trace for a single ray with launch angle -THETALAUNCH
c           for smooth surface. For terrain case, launch angle is THETALAUNCH.
c           Upon reflection the heights of this ray at each output range point
c           RO is then stored in HLIM() for subsequent output of loss values
c           in array MLOSS. This is done so that only loss values that fall
c           within the valid PE solution region are output or passed back in
c           MLOSS.

c Called from: PEINIT

c Routines called: NONE

c PARAMETER LIST:
```

```
c Parameters passed:
c NVROUT = Number of output range points.

c Parameters returned: NONE

      subroutine traceh( nvrout )

      include 'tpem.inc'

      common / trvar / dmdh(mxlvls), zlim, jls, thetalaunch, rlim
      common / rhstps / dr, drout, dzout, dr2, zout(mxzout)
      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                   antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                   hmref
      common / parinit / rv2, refdum(mxlvls), htdum(mxlvls),
     +                   profint(0:maxpts), ht(0:maxpts), is, lvlep

      complex qi
      logical fter

c Define one-line ray trace functions:

      radal( a, b ) = a**2 + 2. * grad * b
      rp( a, b ) = a + b / grad
      ap( a, b ) = a + b * grad
      hp( a, b, c ) = a + ( b**2 - c**2 ) / 2. / grad

      a0 = -thetalaunch
      if( fter ) a0 = thetalaunch
      h0 = antref
      jl = jls
      ro = drout
      ihu = 0
      ihl = 0
      r0 = 0.
      rpe = 0.

c Ray is traced through NROUT output range points.

      do i = 1, nvrout

c Trace until ray reaches output range point RO.

         do while( r0 .lt. ro )

            r1 = ro

            grad = dmdh(jl)
            a1 = ap( a0, r1-r0 )

            if( sign(1.,a0) .ne. sign(1.,a1) ) then
               a1 = 0.
               r1 = rp( r0, a1-a0 )
            end if
            h1 = hp( h0, a1, a0 )

            if(( a1 .le. 0. ) .and. ( h1 .le. htdum(jl) ))  then
               h1 = htdum(jl)
               rad = radal( a0, h1-h0 )
               a1 = -sqrt( rad )
               r1 = rp( r0, a1-a0 )
               jl = jl - 1
               if( jl .eq. 0 ) jl = 1
            elseif(( a1 .ge. 0. ) .and. ( h1 .ge. htdum(jl+1) )) then
               h1 = htdum(jl+1)
               rad = radal( a0, h1-h0 )
               a1 = sqrt( rad )
```

```
            r1 = rp( r0, a1-a0 )
            j1 = j1 + 1
            if( j1 .gt. lvlep ) j1 = lvlep
         end if

         if( r1 .gt. ro ) then
            r1 = ro
            a1 = ap( a0, r1-r0 )
            h1 = hp( h0, a1, a0 )
         end if

         h0 = h1
         r0 = r1
         a0 = a1

         if( h0 .le. 1.e-4 ) then
            a0 = -a0
            rpe = r0
         end if

c If ray has reached ZLIM (maximum output height region) then all heights for
c subsequent output range points will also be at ZLIM - so can exit loop.

         if( h0 .gt. zlim ) then
            ihu = i
            exit
         end if
      end do

      if( ihu .gt. 0 ) exit

      if( a0 .lt. 0. ) hlim(i) = 0.
      if( a0 .ge. 0. ) hlim(i) = h0

      ro = ro + drout

   end do

   if( ihu .gt. 0 ) then
      do i = ihu, nvrout
         hlim(i) = zlim
      end do
   end if

   end

c ****************************** SUBROUTINE XYINIT ************************

c Purpose:  Determines the initial starter field.

c Called from:  PEINIT

c Routines called: ANTPAT

c PARAMETER LIST:

c Parameters passed:
c SV = structure of user-provided system information.
c TR = structure of user-provided terrain data.

c Parameters returned: NONE

   SUBROUTINE xyinit( sv, tr )

   include 'tpem.inc'

   common / arrays / u(0:maxpts), filt(0:maxn4), frsp(0:maxpts),
```

```
     +                    envpr(0:maxpts), ulst(0:maxpts)
      common / miscvar / fnorm, cnst, delp, thetamax, plcnst, qi,
     +                    antref, rpe, hlim(mxrout), slp(mxter), fter,
     +                    hmref
      common / pevar / wl, fko, delz, n, ln, zmax, n34, con, dz2, nm1
      common / impedance / alphav, rav(0:maxpts), rng, rng2, c1, c2,
     +                      rk, c1m, c2m, ig, root

      record / systemvar / sv
      record / terrain / tr

      logical fter

      complex u, frsp, envpr, ulst, qi, root
      complex alphav, rav, rng, rng2, c1, c2, rk, c1m, c2m
      complex refcoef, rterm, dterm, crad, srad

c Reflection coefficient is defaulted to -1 for horizontal polarization.

      refcoef = cmplx( -1., 0. )
      if( sv.polar .eq. 'V' ) call getaln( tr )

      sgain= sqrt( wl ) / zmax

      dtheta = delp / fko
      antko = fko * sv.antht

      DO I=0,N

         pk = float(i) * dtheta
         zpk = pk * antko

c Get antenna pattern factors for the direct and reflected rays.

         call antpat( sv.ipat, pk, FACD )
         call antpat( sv.ipat, -pk, FACR )

c If vertical polarization, then determine reflection coefficient.

         if( sv.polar .eq. 'V' ) then
            ctheta = cos( pk )
            stheta = sin( pk )
            crad = csqrt( rng2 - ctheta**2 )
            srad = rng2 * stheta
            refcoef = (srad - crad) / (srad + crad)
         end if

         rterm = cmplx( cos( zpk ), sin( zpk ) )
         dterm = conjg( rterm )

         u(i) = sgain * ( facd * dterm + refcoef * facr * rterm )

      end do

c Filter upper 1/4 of the field.

      do i = n34, n
         attn = filt(i-n34)
         u(i) = attn*u(i)
      end do

                                    END
```

SINFFT.FOR

```
SUBROUTINE SINFFT ( N, X )
C
C*********************************************************************
C*                                                                   *
C*                                                                   *
C* PURPOSE:   SINFFT replaces the real array X()                     *
C*            by its finite discrete sine transform                  *
C*                                                                   *
C* METHOD :                                                          *
C*                                                                   *
C*     The algorithm is based on a mixed radix (8-4-2) real vector   *
C*     fast Fourier synthesis routine published by Bergland:         *
C*                                                                   *
C*     ( G.D. Bergland, 'A Radix-eight Fast Fourier Transform        *
C*      Subroutine for Real-valued Series,' IEEE Transactions on     *
C*      Audio and Electro-acoustics', vol. AU-17, pp. 138-144, 1969 ) *
C*                                                                   *
C*     and sine and cosine transform algorithms for real series      *
C*     published by Cooley, Lewis, and Welch:                        *
C*                                                                   *
C*     (J.W. COOLEY, P.A.W. LEWIS AND P.D. WELSH, 'The Fast Fourier  *
C*     Transform Algorithm: Programming Considerations in the        *
C*     Calculation of Sine, Cosine and Laplace Transforms',          *
C*     J. SOUND VIB., vol. 12, pp. 315-337, 1970 ).                  *
C*                                                                   *
C*                                                                   *
C* ARGUMENTS:                                                        *
C*                      -- INPUT --                                  *
C*                                                                   *
C*      N...... transform size ( = 2**N )                            *
C*                                                                   *
C*      X().... data array dimensioned 2**N in calling program       *
C*                                                                   *
C*                     -- OUTPUT --                                  *
C*                                                                   *
C*      X().... sine transform                                       *
C*                                                                   *
C*  TABLES:  array     required size                                 *
C*                                                                   *
C*               B          2**N                                     *
C*               JINDX      2**(N-1)                                 *
C*               COSTBL     2**(N-4)                                 *
C*               SINTBL     2**(N-4)                                 *
C*                                                                   *
C*  Sub-programs called:  -                                          *
C*                                                                   *
C*                 R8SYN..... (radix 8 synthesis)                    *
C*                                                                   *
C*********************************************************************
C
      include 'fftsiz.inc'

       INTEGER*4   N
C
       DIMENSION   X(0:*)
       INTEGER*4   NMAX2, NMAX16, NP, NPD2, NPD4
C
       PARAMETER   ( NMAX2  = MAXPTS/2 )
       PARAMETER   ( NMAX16 = MAXPTS/16 )
       DIMENSION   B(MAXPTS), JINDX (NMAX2)
       DIMENSION   SINES (MAXPTS)
       DIMENSION   COSTBL (NMAX16), SINTBL (NMAX16)
C
       SAVE B, COSTBL, JINDX, SINES, SINTBL
       SAVE NSAVE, N4, N8, NP, NPD2, NPD4, NPD16, NPM1
```

```
C
C
      DOUBLE PRECISION ARG, DT, PI
      DATA NSAVE  / 0 /
      DATA PI     / 3.1415 92653 58979 32D0 /
C
C----------------------------------------------------------------------+
C

      IF ( N .NE. NSAVE ) THEN
C                                  compute constants and construct tables
         NSAVE = N
         N8 = NSAVE / 3
         N4 = NSAVE - 3 * N8 - 1
         NP = 2**N
         NPD2  = NP / 2
         NPD4  = NP / 4
         NPD16 = NP / 16
         NPM1  = NP - 1
C                                       build reciprical sine table
         DT    = PI / FLOAT ( NP )
         DO 10 J = 1, NPM1
            ARG = DT * J
            SINES ( J ) = (0.5D0 / SIN ( ARG ))
   10    CONTINUE
C                                  construct bit reversed subscript table
         J1 = 0
         DO 30 J = 1, NPD2 - 1
            J2 = NPD2
   20       CONTINUE
            IF ( IAND ( J1, J2 ) .NE. 0 ) THEN
               J1 = IABS ( J1 - J2 )
               J2 = J2 / 2
               GO TO 20
            ENDIF
            J1 = J1 + J2
            JINDX ( J ) = J1
   30    CONTINUE
C
C                           form the trig tables for the radix-8 passes;
C                           tables are stored in bit reversed order.
         J1 = 0
         DO 50 J = 1, NPD16 - 1
            J2 = NPD16
   40       CONTINUE
            IF ( IAND ( J1, J2 ) .NE. 0 ) THEN
               J1 = IABS ( J1 - J2 )
               J2 = J2 / 2
               GO TO 40
            ENDIF
            J1  = J1 + J2
            ARG = DT * FLOAT (J1)
            COSTBL ( J ) =  COS ( ARG )
            SINTBL ( J ) = -SIN ( ARG )
   50    CONTINUE
C
      ENDIF
C
C                      ***  form the input Fourier coefficients ***
C
C                                                        sine transform
         B ( 1 ) = -2. * X ( 1 )
         B ( 2 ) =  2. * X ( NPM1 )
         J1 = 0
         DO 110 J = 3, NPM1, 2
            J1 = J1 + 1
            J2 = JINDX ( J1 )
```

```
            B ( J )     = X ( J2 - 1 ) - X ( J2 + 1 )
            B ( J + 1 ) = X ( NP-J2 )
  110     CONTINUE
C
C                    ************************************
C                    *                                  *
C                    *    Begin Fast Fourier Synthesis  *
C                    *                                  *
C                    ************************************
C
      IF ( N8 .NE. 0 ) THEN
C                                                radix-8 iterations
         INTT = 1
         NT = NPD16
         DO 130 J = 1, N8
            J1 =  1 + INTT
            J2 = J1 + INTT
            J3 = J2 + INTT
            J4 = J3 + INTT
            J5 = J4 + INTT
            J6 = J5 + INTT
            J7 = J6 + INTT
C***
            CALL R8SYN (INTT, NT, COSTBL, SINTBL, B(1), B(J1), B(J2),
     *                     B(J3), B(J4), B(J5), B(J6), B(J7)  )
C***
            NT = NT / 8
            INTT = 8 * INTT
  130     CONTINUE
      ENDIF
C                                                radix-4 iteration
      IF ( N4 .GT. 0 ) THEN
         J1 = NPD4
         J2 = 2*NPD4
         J3 = 3*NPD4
         DO 140 J = 1, NPD4
            T0 = B(J) + B(J + J1)
            T1 = B(J) - B(J + J1)
            T2 = 2. * B(J + J2)
            T3 = 2. * B(J + J3)
            B(J)      = T0 + T2
            B(J + J2) = T0 - T2
            B(J + J1) = T1 + T3
            B(J + J3) = T1 - T3
  140     CONTINUE
C
      ELSE IF ( N4 .EQ. 0 ) THEN
C                                                radix-2 iteration
         K = NPD2
         DO 150 J = 1, NPD2
            K    = K + 1
            T    = B(J) + B (K)
            B(K) = B(J) - B (K)
            B(J) = T
  150     CONTINUE
      ENDIF
C
C                    ***********************
C                    *                     *
C                    *   Form Transform    *
C                    *                     *
C                    ***********************
C
C
C                                                sine transform
         J1 = NP
         DO 160 J = 1, NPM1
```

```
      X(J) = .25*(( B(J+1) + B(J1)) * SINES(J) - B(J+1) + B(J1))
        J1 = J1 - 1
160      CONTINUE

      RETURN
      END
C
C
      SUBROUTINE R8SYN ( INTT, NT, COSTBL, SINTBL, B0, B1, B2, B3,
     *                   B4, B5, B6, B7 )
C
C***********************************************************************
C
C  PURPOSE:    Radix-8 synthesis subroutine used by mixed radix driver.
C
C
C
C***********************************************************************
C
C
      DIMENSION   COSTBL(*), SINTBL(*)
      DIMENSION B0(*), B1(*), B2(*), B3(*), B4(*), B5(*), B6(*), B7(*)
C
C
C           ///     Local variables     ///
C
C
      DOUBLE PRECISION C1, C2, C3, C4, C5, C6, C7
      DOUBLE PRECISION S1, S2, S3, S4, S5, S6, S7
      DOUBLE PRECISION CPI4, CPI8, R2, SPI8
C
      SAVE  CPI4, CPI8,  R2, SPI8
C
      DATA  R2     / 1.41421 35623 7310D+0 /,
     *      CPI4   / 0.70710 67811 8655D+0 /,
     *      CPI8   / 0.92387 95325 1129D+0 /,
     *      SPI8   / 0.38268 34323 6509D+0 /
C
C
C---------------------------------------------------------------------+
C
     JT = 0
     JL = 2
     JR = 2
     JI = 3
     INT8 = 8 * INTT
C
     DO 60 K = 1, INTT
       T0 = B0(K) + B1(K)
       T1 = B0(K) - B1(K)
       T2 = B2(K) + B2(K)
       T3 = B3(K) + B3(K)
       T4 = B4(K) + B6(K)
       T5 = B4(K) - B6(K)
       T6 = B7(K) - B5(K)
       T7 = B7(K) + B5(K)
       T8 = R2 * (T7 - T5)
       T5 = R2 * (T7 + T5)
       TT0 = T0 + T2
       T2  = T0 - T2
       TT1 = T1 + T3
       T3  = T1 - T3
       T4  = T4 + T4
       T6  = T6 + T6
C
       B0(K) = TT0 + T4
       B4(K) = TT0 - T4
```

```
      B1(K) = TT1 + T5
      B5(K) = TT1 - T5
      B2(K) = T2 + T6
      B6(K) = T2 - T6
      B3(K) = T3 + T8
      B7(K) = T3 - T8
   60 CONTINUE
C
      IF ( NT .EQ. 0 )                    RETURN
C
      K0 = INT8 + 1
      KLAST = INT8 + INTT
C
      DO 70 K = K0, KLAST
         T1 = B0(K) + B6(K)
         T3 = B0(K) - B6(K)
         T2 = B7(K) - B1(K)
         T4 = B7(K) + B1(K)
         T5 = B2(K) + B4(K)
         T7 = B2(K) - B4(K)
         T6 = B5(K) - B3(K)
         T8 = B5(K) + B3(K)
C
         B0(K) = (T1 + T5) + (T1 + T5)
         B4(K) = (T2 + T6) + (T2 + T6)
         T5    = T1 - T5
         T6    = T2 - T6
         B2(K) = R2 * (T6 + T5)
         B6(K) = R2 * (T6 - T5)
         T1    = T3 * CPI8 + T4 * SPI8
         T2    = T4 * CPI8 - T3 * SPI8
         T3    = T8 * CPI8 - T7 * SPI8
         T4    =  - T7 * CPI8 - T8 * SPI8
         B1(K) = (T1 + T3) + (T1 + T3)
         B5(K) = (T2 + T4) + (T2 + T4)
         T3    = T1 - T3
         T4    = T2 - T4
         B3(K) = R2 * (T4 + T3)
         B7(K) = R2 * (T4 - T3)
   70  CONTINUE
C
      DO 90 JT = 1, NT-1
       C1 = COSTBL(JT)
       S1 = SINTBL(JT)
       C2 = C1 * C1 - S1 * S1
       S2 = C1 * S1 + C1 * S1
       C3 = C1 * C2 - S1 * S2
       S3 = C2 * S1 + S2 * C1
       C4 = C2 * C2 - S2 * S2
       S4 = C2 * S2 + C2 * S2
       C5 = C2 * C3 - S2 * S3
       S5 = C3 * S2 + S3 * C2
       C6 = C3 * C3 - S3 * S3
       S6 = C3 * S3 + C3 * S3
       C7 = C3 * C4 - S3 * S4
       S7 = C4 * S3 + S4 * C3
C
       K  = JI * INT8
       J0 = JR * INT8 + 1
       JLAST = J0 + INTT - 1
C
       DO 80 J = J0, JLAST
C
          K   = K + 1
          TR0 = B0(J) + B6(K)
          TR1 = B0(J) - B6(K)
          TI0 = B7(K) - B1(J)
```

```
            TI1 = B7(K)  + B1(J)
            TR2 = B4(K)  + B2(J)
            TI3 = B4(K)  - B2(J)
            TI2 = B5(K)  - B3(J)
            TR3 = B5(K)  + B3(J)
            TR4 = B4(J)  + B2(K)
            T0  = B4(J)  - B2(K)
            TI4 = B3(K)  - B5(J)
            T1  = B3(K)  + B5(J)
            TR5 = CPI4 *  (T1 + T0)
            TI5 = CPI4 *  (T1 - T0)
            TR6 = B6(J)  + B0(K)
            T0  = B6(J)  - B0(K)
            TI6 = B1(K)  - B7(J)
            T1  = B1(K)  + B7(J)
            TR7  =   - CPI4 *  (T0 - T1)
            TI7  =   - CPI4 *  (T0 + T1)
            T0   = TR0 + TR2
            TR2  = TR0 - TR2
            T1   = TI0 + TI2
            TI2  = TI0 - TI2
            T2   = TR1 + TR3
            TR3  = TR1 - TR3
            T3   = TI1 + TI3
            TI3  = TI1 - TI3
            T5   = TI4 + TI6
            TTR6 = TI4 - TI6
            TI6  = TR6 - TR4
            T4   = TR4 + TR6
            T7   = TI5 + TI7
            TTR7 = TI5 - TI7
            TI7  = TR7 - TR5
            T6   = TR5 + TR7
C
            B0(J)  = T0 + T4
            B0(K)  = T1 + T5
            B4(J)  = C4 *  (T0 - T4)     - S4 *  (T1 - T5)
            B4(K)  = C4 *  (T1 - T5)     + S4 *  (T0 - T4)
C
            B1(J)  = C1 *  (T2 + T6)     - S1 *  (T3 + T7)
            B1(K)  = C1 *  (T3 + T7)     + S1 *  (T2 + T6)
            B5(J)  = C5 *  (T2 - T6)     - S5 *  (T3 - T7)
            B5(K)  = C5 *  (T3 - T7)     + S5 *  (T2 - T6)
C
            B2(J)  = C2 *  (TR2 + TTR6)  - S2 *  (TI2 + TI6)
            B2(K)  = C2 *  (TI2 + TI6)   + S2 *  (TR2 + TTR6)
            B6(J)  = C6 *  (TR2 - TTR6)  - S6 *  (TI2 - TI6)
            B6(K)  = C6 *  (TI2 - TI6)   + S6 *  (TR2 - TTR6)
C
            B3(J)  = C3 *  (TR3 + TTR7)  - S3 *  (TI3 + TI7)
            B3(K)  = C3 *  (TI3 + TI7)   + S3 *  (TR3 + TTR7)
            B7(J)  = C7 *  (TR3 - TTR7)  - S7 *  (TI3 - TI7)
            B7(K)  = C7 *  (TI3 - TI7)   + S7 *  (TR3 - TTR7)
C
   80   CONTINUE
C
        JR = JR + 2
        JI = JI - 2
        IF ( JI .GT. JL) GOTO 90
        JI = JR + JR - 1
        JL = JR
   90 CONTINUE
C
      RETURN
      END
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
February 1996

3. REPORT TYPE AND DATES COVERED
Final: February 1996

4. TITLE AND SUBTITLE
TERRAIN PARABOLIC EQUATION MODEL (TPEM) VERSION 1.5 USER'S MANUAL

5. FUNDING NUMBERS
PE: 0602435N
WU: 54–MPB01
AN: DN302214

6. AUTHOR(S)
A. E. Barrios

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Command, Control and Ocean Surveillance Center
RDT&E Division
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER
TD 2898

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217–5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document describes the Terrain Parabolic Equation Model (TPEM) Version 1.5, its operation, and the format required for the environmental and system input files. TPEM 1.5 calculates and plots propagation loss in dB on a height vs. range display. It allows for range-dependent refractivity environments and variable terrain. TPEM 1.5 is based on methods and source code originally developed by Fred Tappert, from the University of Miami, for propagation over a smooth surface. It is a pure PE model based on the split-step Fourier method.

14. SUBJECT TERMS
Mission Area: Communications
propagation loss
terrain parabolic equation model
atmospheric refractivity

15. NUMBER OF PAGES
73

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
A. E. Barrios	(619) 553–1429	Code 543

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0271	Archive/Stock	(6)
Code 0274	Library	(2)
Code 88	J. H. Richter	(1)
Code 883	R. A. Paulus	(1)
Code 883	A. E. Barrios	(30)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363–5100

Center for Naval Analyses
Alexandria, VA 22302–0268

Navy Acquisition, Research and Development
Information Center (NARDIC)
Arlington, VA 22244–5114

GIDEP Operations Center
Corona, CA 91718–8000